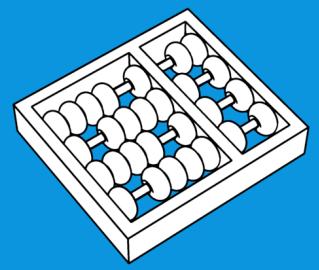
Python: primeiros passos

MC102 - Algoritmos e Programação de Computadores

Santiago Valdés Ravelo
https://ic.unicamp.br/~santiago/
ravelo@unicamp.br

UNICAMP



“Python é o canivete suíço para programar, tem todo o que você precisa.”

Anônimo.

Dúvidas da aula anterior

Dúvidas selecionadas

- Fiquei com dúvida quanto a diferença entre memória RAM, memória cache e acumulador (?)

Dúvidas selecionadas

- Fiquei com dúvida quanto a diferença entre memória RAM, memória cache e acumulador (?)
- Arquitetura de Von Neumann: Não entendi o que é o Apontador e o Acumulador. Em um computador real, seriam componentes(hardware) localizados dentro do processador (como na imagem), ou seria um algoritmo de software (como um código)?

Dúvidas selecionadas

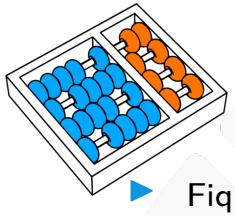
- Fiquei com dúvida quanto a diferença entre memória RAM, memória cache e acumulador (?)
- Arquitetura de Von Neumann: Não entendi o que é o Apontador e o Acumulador. Em um computador real, seriam componentes(hardware) localizados dentro do processador (como na imagem), ou seria um algoritmo de software (como um código)?
- Como o acumulador funciona como um armazenamento temporário de informações, eu poderia associar ele ao papel de uma variável?

Dúvidas selecionadas

- Fiquei com dúvida quanto a diferença entre memória RAM, memória cache e acumulador (?)
- Arquitetura de Von Neumann: Não entendi o que é o Apontador e o Acumulador. Em um computador real, seriam componentes(hardware) localizados dentro do processador (como na imagem), ou seria um algoritmo de software (como um código)?
- Como o acumulador funciona como um armazenamento temporário de informações, eu poderia associar ele ao papel de uma variável?
- Como os componentes internos do celuar são basicamente os mesmos de um computador, o celular pode ser considerado um compiutador compactado?

Dúvidas selecionadas

- Fiquei com dúvida quanto a diferença entre memória RAM, memória cache e acumulador (?)
- Arquitetura de Von Neumann: Não entendi o que é o Apontador e o Acumulador. Em um computador real, seriam componentes(hardware) localizados dentro do processador (como na imagem), ou seria um algoritmo de software (como um código)?
- Como o acumulador funciona como um armazenamento temporário de informações, eu poderia associar ele ao papel de uma variável?
- Como os componentes internos do celuar são basicamente os mesmos de um computador, o celular pode ser considerado um compiutador compactado?
- Eu fiquei com uma dúvida em relação à necessidade de algumas linguagens precisarem de compiladores, como python, e outras, como C e C++ não precisarem. Qual a razão disso?



Dúvidas selecionadas

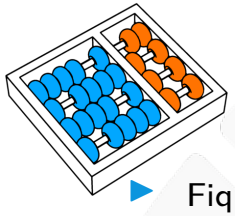
- Fiquei com dúvida quanto a diferença entre memória RAM, memória cache e acumulador (?)
- Arquitetura de Von Neumann: Não entendi o que é o Apontador e o Acumulador. Em um computador real, seriam componentes(hardware) localizados dentro do processador (como na imagem), ou seria um algoritmo de software (como um código)?
- Como o acumulador funciona como um armazenamento temporário de informações, eu poderia associar ele ao papel de uma variável?
- Como os componentes internos do celuar são basicamente os mesmos de um computador, o celular pode ser considerado um compiutador compactado?
- Eu fiquei com uma dúvida em relação à necessidade de algumas linguagens precisarem de compiladores, como python, e outras, como C e C++ não precisarem. Qual a razão disso?
- O quão importante é entender o funcionamento de baixo nível de um computador (aspectos como a arquitetura, que foram abordados na aula) para desenvolver um programa em alto nível, em python, por exemplo?

Dúvidas selecionadas

- Fiquei com dúvida quanto a diferença entre memória RAM, memória cache e acumulador (?)
- Arquitetura de Von Neumann: Não entendi o que é o Apontador e o Acumulador. Em um computador real, seriam componentes(hardware) localizados dentro do processador (como na imagem), ou seria um algoritmo de software (como um código)?
- Como o acumulador funciona como um armazenamento temporário de informações, eu poderia associar ele ao papel de uma variável?
- Como os componentes internos do celuar são basicamente os mesmos de um computador, o celular pode ser considerado um compiutador compactado?
- Eu fiquei com uma dúvida em relação à necessidade de algumas linguagens precisarem de compiladores, como python, e outras, como C e C++ não precisarem. Qual a razão disso?
- O quão importante é entender o funcionamento de baixo nível de um computador (aspectos como a arquitetura, que foram abordados na aula) para desenvolver um programa em alto nível, em python, por exemplo?
- Não sei se isso foi explicado semana passada, mas, por que MC102 será em Python? Fiquei curiosa, pois soube que, antigamente, era em C.

Dúvidas selecionadas

- Fiquei com dúvida quanto a diferença entre memória RAM, memória cache e acumulador (?)
- Arquitetura de Von Neumann: Não entendi o que é o Apontador e o Acumulador. Em um computador real, seriam componentes(hardware) localizados dentro do processador (como na imagem), ou seria um algoritmo de software (como um código)?
- Como o acumulador funciona como um armazenamento temporário de informações, eu poderia associar ele ao papel de uma variável?
- Como os componentes internos do celuar são basicamente os mesmos de um computador, o celular pode ser considerado um compiutador compactado?
- Eu fiquei com uma dúvida em relação à necessidade de algumas linguagens precisarem de compiladores, como python, e outras, como C e C++ não precisarem. Qual a razão disso?
- O quão importante é entender o funcionamento de baixo nível de um computador (aspectos como a arquitetura, que foram abordados na aula) para desenvolver um programa em alto nível, em python, por exemplo?
- Não sei se isso foi explicado semana passada, mas, por que MC102 será em Python? Fiquei curiosa, pois soube que, antigamente, era em C.
- como um programa grande é executado em uma memória RAM? por exemplo: como um jogo de 100GB é executado em um computador de 8GB ou 16GB?



Dúvidas selecionadas

- Fiquei com dúvida quanto a diferença entre memória RAM, memória cache e acumulador (?)
- Arquitetura de Von Neumann: Não entendi o que é o Apontador e o Acumulador. Em um computador real, seriam componentes(hardware) localizados dentro do processador (como na imagem), ou seria um algoritmo de software (como um código)?
- Como o acumulador funciona como um armazenamento temporário de informações, eu poderia associar ele ao papel de uma variável?
- Como os componentes internos do celuar são basicamente os mesmos de um computador, o celular pode ser considerado um compiutador compactado?
- Eu fiquei com uma dúvida em relação à necessidade de algumas linguagens precisarem de compiladores, como python, e outras, como C e C++ não precisarem. Qual a razão disso?
- O quão importante é entender o funcionamento de baixo nível de um computador (aspectos como a arquitetura, que foram abordados na aula) para desenvolver um programa em alto nível, em python, por exemplo?
- Não sei se isso foi explicado semana passada, mas, por que MC102 será em Python? Fiquei curiosa, pois soube que, antigamente, era em C.
- como um programa grande é executado em uma memória RAM? por exemplo: como um jogo de 100GB é executado em um computador de 8GB ou 16GB?
- Pq precisa transferir os dados da memoria para o acumulador? pelo falo dele ser integrado ao processador?

Dúvidas selecionadas

- Fiquei com dúvida quanto a diferença entre memória RAM, memória cache e acumulador (?)
- Arquitetura de Von Neumann: Não entendi o que é o Apontador e o Acumulador. Em um computador real, seriam componentes(hardware) localizados dentro do processador (como na imagem), ou seria um algoritmo de software (como um código)?
- Como o acumulador funciona como um armazenamento temporário de informações, eu poderia associar ele ao papel de uma variável?
- Como os componentes internos do celuar são basicamente os mesmos de um computador, o celular pode ser considerado um compiutador compactado?
- Eu fiquei com uma dúvida em relação à necessidade de algumas linguagens precisarem de compiladores, como python, e outras, como C e C++ não precisarem. Qual a razão disso?
- O quão importante é entender o funcionamento de baixo nível de um computador (aspectos como a arquitetura, que foram abordados na aula) para desenvolver um programa em alto nível, em python, por exemplo?
- Não sei se isso foi explicado semana passada, mas, por que MC102 será em Python? Fiquei curiosa, pois soube que, antigamente, era em C.
- como um programa grande é executado em uma memória RAM? por exemplo: como um jogo de 100GB é executado em um computador de 8GB ou 16GB?
- Pq precisa transferir os dados da memoria para o acumulador? pelo falo dele ser integrado ao processador?
- Por que a pasta de arquivos temporários de um aplicativo no celular tem o nome de Cache mesmo que se assemelhe mais a uma pasta de arquivos temporários, sem nenhuma semelhança com a "verdadeira"memória Cache?

Dúvidas selecionadas

- Fiquei com dúvida quanto a diferença entre memória RAM, memória cache e acumulador (?)
- Arquitetura de Von Neumann: Não entendi o que é o Apontador e o Acumulador. Em um computador real, seriam componentes(hardware) localizados dentro do processador (como na imagem), ou seria um algoritmo de software (como um código)?
- Como o acumulador funciona como um armazenamento temporário de informações, eu poderia associar ele ao papel de uma variável?
- Como os componentes internos do celuar são basicamente os mesmos de um computador, o celular pode ser considerado um compiutador compactado?
- Eu fiquei com uma dúvida em relação à necessidade de algumas linguagens precisarem de compiladores, como python, e outras, como C e C++ não precisarem. Qual a razão disso?
- O quão importante é entender o funcionamento de baixo nível de um computador (aspectos como a arquitetura, que foram abordados na aula) para desenvolver um programa em alto nível, em python, por exemplo?
- Não sei se isso foi explicado semana passada, mas, por que MC102 será em Python? Fiquei curiosa, pois soube que, antigamente, era em C.
- como um programa grande é executado em uma memória RAM? por exemplo: como um jogo de 100GB é executado em um computador de 8GB ou 16GB?
- Pq precisa transferir os dados da memoria para o acumulador? pelo falo dele ser integrado ao processador?
- Por que a pasta de arquivos temporários de um aplicativo no celular tem o nome de Cache mesmo que se assemelhe mais a uma pasta de arquivos temporários, sem nenhuma semelhança com a "verdadeira"memória Cache?
- Os computadores "normais"de hoje são feitos apenas com os componentes básicos que você mostrou (e mais algumas coisas extras) ou possuem muito mais coisa e são muito mais complexos que isso?

Dúvidas selecionadas

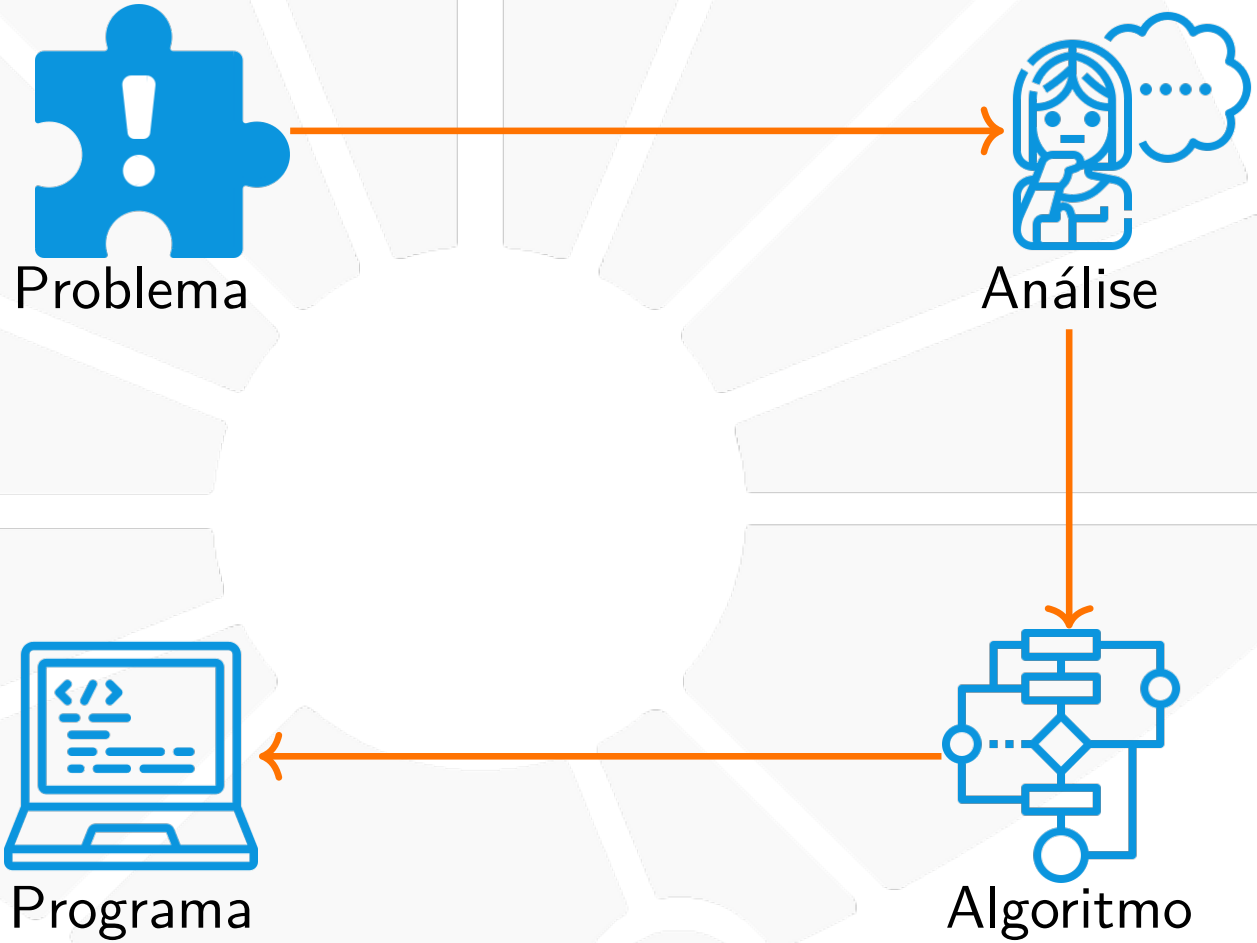
- Fiquei com dúvida quanto a diferença entre memória RAM, memória cache e acumulador (?)
- Arquitetura de Von Neumann: Não entendi o que é o Apontador e o Acumulador. Em um computador real, seriam componentes(hardware) localizados dentro do processador (como na imagem), ou seria um algoritmo de software (como um código)?
- Como o acumulador funciona como um armazenamento temporário de informações, eu poderia associar ele ao papel de uma variável?
- Como os componentes internos do celuar são basicamente os mesmos de um computador, o celular pode ser considerado um compiutador compactado?
- Eu fiquei com uma dúvida em relação à necessidade de algumas linguagens precisarem de compiladores, como python, e outras, como C e C++ não precisarem. Qual a razão disso?
- O quão importante é entender o funcionamento de baixo nível de um computador (aspectos como a arquitetura, que foram abordados na aula) para desenvolver um programa em alto nível, em python, por exemplo?
- Não sei se isso foi explicado semana passada, mas, por que MC102 será em Python? Fiquei curiosa, pois soube que, antigamente, era em C.
- como um programa grande é executado em uma memória RAM? por exemplo: como um jogo de 100GB é executado em um computador de 8GB ou 16GB?
- Pq precisa transferir os dados da memoria para o acumulador? pelo falo dele ser integrado ao processador?
- Por que a pasta de arquivos temporários de um aplicativo no celular tem o nome de Cache mesmo que se assemelhe mais a uma pasta de arquivos temporários, sem nenhuma semelhança com a "verdadeira"memória Cache?
- Os computadores "normais"de hoje são feitos apenas com os componentes básicos que você mostrou (e mais algumas coisas extras) ou possuem muito mais coisa e são muito mais complexos que isso?
- Como a GPU lida com as instruções e memória em comparação com a CPU? Além disso, existe diferença na forma como os dados são enviados para o dispositivo de saída tela em um sistema com GPU dedicada, em um com GPU integrada à CPU e em um sem GPU?

Dúvidas selecionadas

- Fiquei com dúvida quanto a diferença entre memória RAM, memória cache e acumulador (?)
- Arquitetura de Von Neumann: Não entendi o que é o Apontador e o Acumulador. Em um computador real, seriam componentes(hardware) localizados dentro do processador (como na imagem), ou seria um algoritmo de software (como um código)?
- Como o acumulador funciona como um armazenamento temporário de informações, eu poderia associar ele ao papel de uma variável?
- Como os componentes internos do celuar são basicamente os mesmos de um computador, o celular pode ser considerado um compiutador compactado?
- Eu fiquei com uma dúvida em relação à necessidade de algumas linguagens precisarem de compiladores, como python, e outras, como C e C++ não precisarem. Qual a razão disso?
- O quão importante é entender o funcionamento de baixo nível de um computador (aspectos como a arquitetura, que foram abordados na aula) para desenvolver um programa em alto nível, em python, por exemplo?
- Não sei se isso foi explicado semana passada, mas, por que MC102 será em Python? Fiquei curiosa, pois soube que, antigamente, era em C.
- como um programa grande é executado em uma memória RAM? por exemplo: como um jogo de 100GB é executado em um computador de 8GB ou 16GB?
- Pq precisa transferir os dados da memoria para o acumulador? pelo falo dele ser integrado ao processador?
- Por que a pasta de arquivos temporários de um aplicativo no celular tem o nome de Cache mesmo que se assemelhe mais a uma pasta de arquivos temporários, sem nenhuma semelhança com a "verdadeira"memória Cache?
- Os computadores "normais"de hoje são feitos apenas com os componentes básicos que você mostrou (e mais algumas coisas extras) ou possuem muito mais coisa e são muito mais complexos que isso?
- Como a GPU lida com as instruções e memória em comparação com a CPU? Além disso, existe diferença na forma como os dados são enviados para o dispositivo de saída tela em um sistema com GPU dedicada, em um com GPU integrada à CPU e em um sem GPU?
- Por que a memoria RAM funciona melhor em dual chanel, ou seja, 2 pentes de 8gb RAM é melhor que 1 pente de 16gb RAM?

Criação de programas

Solução de um problema



Problemas

- Calcular a quantidade de azulejos (quadrados) necessários para cobrir uma parede retangular.

Problemas

- Calcular a quantidade de azulejos (quadrados) necessários para cobrir uma parede retangular.
- Determinar o último dígito de um número inteiro.

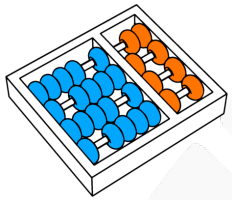
Definição dos problemas

- Calcular a quantidade de azulejos (quadrados) necessários para cobrir uma parede retangular.

Definição dos problemas

- Calcular a quantidade de azulejos (quadrados) necessários para cobrir uma parede retangular.

Entrada três números positivos: a largura dos azulejos, a largura e a altura da parede.

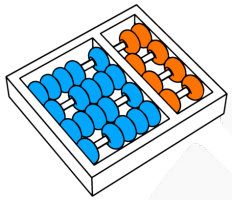


Definição dos problemas

- Calcular a quantidade de azulejos (quadrados) necessários para cobrir uma parede retangular.

Entrada três números positivos: a largura dos azulejos, a largura e a altura da parede.

Saída um número inteiro positivo: o menor número de azulejos cuja soma das áreas é maior ou igual que a área da parede.



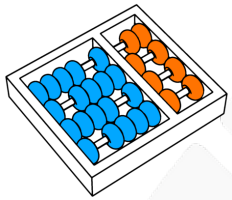
Definição dos problemas

- Calcular a quantidade de azulejos (quadrados) necessários para cobrir uma parede retangular.

  Entrada três números positivos: a largura dos azulejos, a largura e a altura da parede.

  Saída um número inteiro positivo: o menor número de azulejos cuja soma das áreas é maior ou igual que a área da parede.

- Determinar o último dígito de um número inteiro.



Definição dos problemas

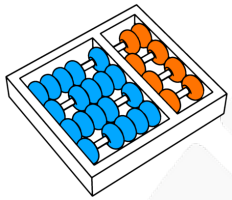
- Calcular a quantidade de azulejos (quadrados) necessários para cobrir uma parede retangular.

Entrada três números positivos: a largura dos azulejos, a largura e a altura da parede.

Saída um número inteiro positivo: o menor número de azulejos cuja soma das áreas é maior ou igual que a área da parede.

- Determinar o último dígito de um número inteiro.

Entrada um número inteiro.



Definição dos problemas

- Calcular a quantidade de azulejos (quadrados) necessários para cobrir uma parede retangular.

Entrada três números positivos: a largura dos azulejos, a largura e a altura da parede.

Saída um número inteiro positivo: o menor número de azulejos cuja soma das áreas é maior ou igual que a área da parede.

- Determinar o último dígito de um número inteiro.

Entrada um número inteiro.

Saída um número inteiro entre 0 e 9 igual ao último dígito do número da entrada.

Algoritmo para um problema. Exemplo

Algoritmo: AZULEJOS-PAREDE

1 **leia** *largura_azulejo*
2 **leia** *largura_parede*
3 **leia** *altura_parede*
4 $area_azulejo \leftarrow largura_azulejo^2$
5 $area_parede \leftarrow largura_parede \cdot altura_parede$
6 $quantidade \leftarrow \lceil \frac{area_parede}{area_azulejo} \rceil$
7 **escreva** *quantidade*

Algoritmo para um problema. Exemplo

Algoritmo: ULTIMO-DIGITO

1 **leia** *numero*
2 $digito \leftarrow numero(mod\ 10)$
3 **escreva** *digito*

Pergunta

Como programar um algoritmo?

Python

Características

- Alto nível.

Características

- Alto nível.
- Multi-paradigma.

Características

- Alto nível.
- Multi-paradigma.
- Multi-propósito.

Características

- Alto nível.
- Multi-paradigma.
- Multi-propósito.
- Multiplataforma.

Características

- Alto nível.
- Multi-paradigma.
- Multi-propósito.
- Multiplataforma.
- Simples.

Características

- Alto nível.
- Multi-paradigma.
- Multi-propósito.
- Multiplataforma.
- Simples.
- Software livre.

Características

- Alto nível.
- Multi-paradigma.
- Multi-propósito.
- Multiplataforma.
- Simples.
- Software livre.
- Muito utilizado.

Onde encontrar?

- Para windows: **https://www.howtogeek.com/197947/how-to-install-python-on-windows/**

Onde encontrar?

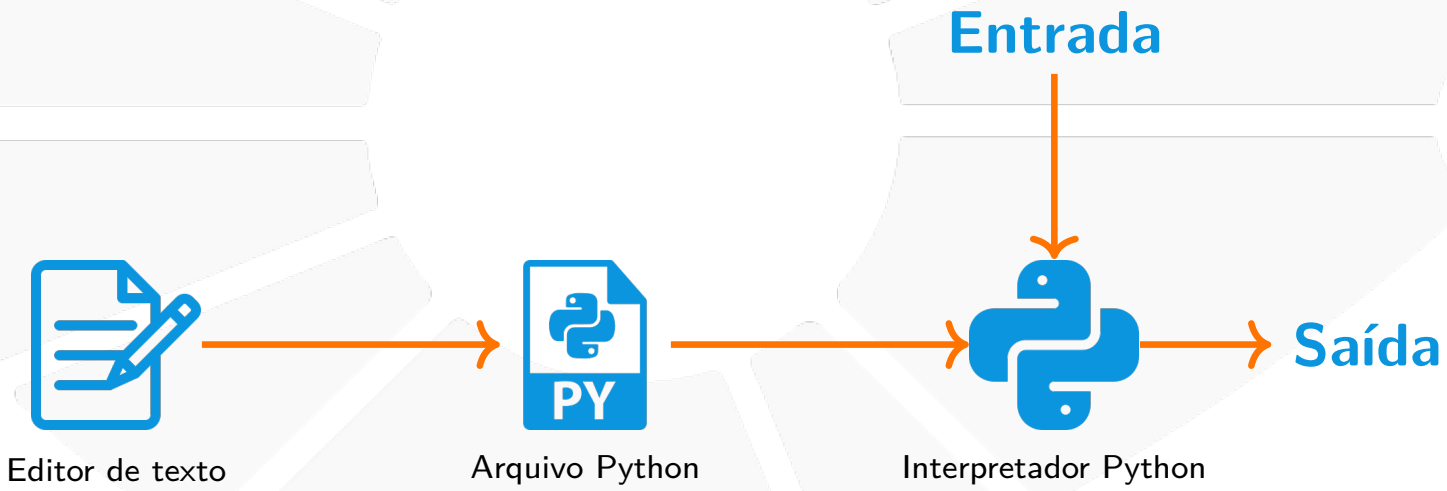
- Para windows: **https://www.howtogeek.com/197947/how-to-install-python-on-windows/**
- Para linux: **https://python.org.br/instalacao-linux/**

Onde encontrar?

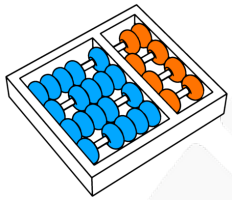
- Para windows: **https://www.howtogeek.com/197947/how-to-install-python-on-windows/**
- Para linux: **https://python.org.br/instalacao-linux/**
- Online : **https://www.tutorialspoint.com/ipython_terminal_online.php**

Modos de uso

- **Interativo**. Python possui um editor interativo que executa instruções na medida em que os comandos são digitados.
- **Não interativo**.



Operações aritméticas básicas



Operações aritméticas básicas

Algumas operações que podemos fazer:

Operações aritméticas básicas

Algumas operações que podemos fazer:

- Soma: $+$.

Operações aritméticas básicas

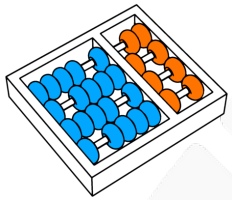
Algumas operações que podemos fazer:

- Soma: $+$.
- Subtração: $-$.

Operações aritméticas básicas

Algumas operações que podemos fazer:

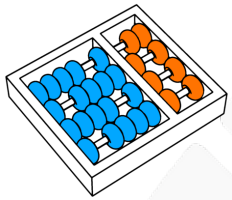
- Soma: $+$.
- Subtração: $-$.
- Multiplicação: $*$.



Operações aritméticas básicas

Algumas operações que podemos fazer:

- Soma: $+$.
- Subtração: $-$.
- Multiplicação: $*$.
- Divisão: $/$.



Operações aritméticas básicas

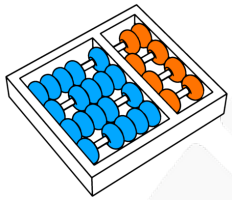
Algumas operações que podemos fazer:

- Soma: $+$.
- Subtração: $-$.
- Multiplicação: $*$.
- Divisão: $/$.
- Exponenciação: $**$.

Operações aritméticas básicas

Algumas operações que podemos fazer:

- Soma: $+$.
- Subtração: $-$.
- Multiplicação: $*$.
- Divisão: $/$.
- Exponenciação: $**$.

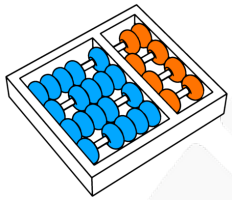


Operações aritméticas básicas

Algumas operações que podemos fazer:

- Soma: $+$.
- Subtração: $-$.
- Multiplicação: $*$.
- Divisão: $/$.
- Exponenciação: $**$.

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

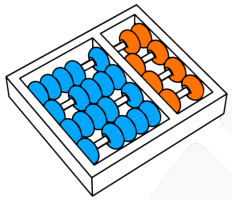


Operações aritméticas básicas

Algumas operações que podemos fazer:

- Soma: $+$.
- Subtração: $-$.
- Multiplicação: $*$.
- Divisão: $/$.
- Exponenciação: $**$.

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

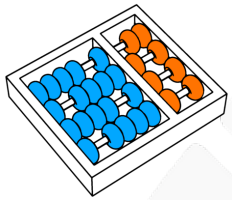


Operações aritméticas básicas

Algumas operações que podemos fazer:

- Soma: $+$.
- Subtração: $-$.
- Multiplicação: $*$.
- Divisão: $/$.
- Exponenciação: $**$.

```
>>> 3.2 + 7
10.2
```

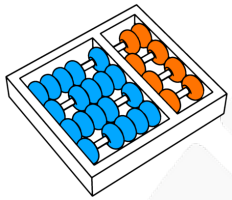


Operações aritméticas básicas

Algumas operações que podemos fazer:

- Soma: $+$.
- Subtração: $-$.
- Multiplicação: $*$.
- Divisão: $/$.
- Exponenciação: $**$.

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

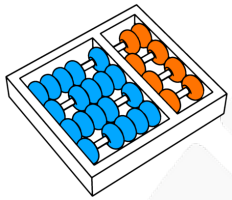


Operações aritméticas básicas

Algumas operações que podemos fazer:

- Soma: $+$.
- Subtração: $-$.
- Multiplicação: $*$.
- Divisão: $/$.
- Exponenciação: $**$.

```
>>> 3.2 + 7
10.2
>>> 5.3 - -2
7.3
```

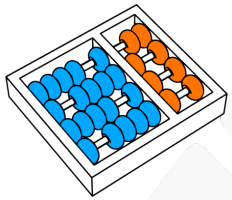


Operações aritméticas básicas

Algumas operações que podemos fazer:

- Soma: $+$.
- Subtração: $-$.
- Multiplicação: $*$.
- Divisão: $/$.
- Exponenciação: $**$.

```
>>> 3.2 + 7
10.2
>>> 5.3 - -2
7.3
>>> 2 * 7.5
```

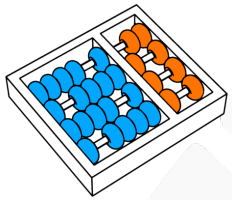


Operações aritméticas básicas

Algumas operações que podemos fazer:

- Soma: $+$.
- Subtração: $-$.
- Multiplicação: $*$.
- Divisão: $/$.
- Exponenciação: $**$.

```
>>> 3.2 + 7
10.2
>>> 5.3 - -2
7.3
>>> 2 * 7.5
15.0
```

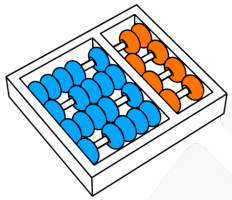


Operações aritméticas básicas

Algumas operações que podemos fazer:

- Soma: $+$.
- Subtração: $-$.
- Multiplicação: $*$.
- Divisão: $/$.
- Exponenciação: $**$.

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

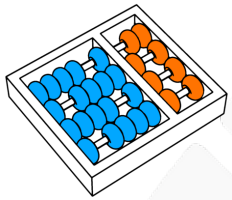


Operações aritméticas básicas

Algumas operações que podemos fazer:

- Soma: $+$.
- Subtração: $-$.
- Multiplicação: $*$.
- Divisão: $/$.
- Exponenciação: $**$.

```
>>> 3.2 + 7
10.2
>>> 5.3 - -2
7.3
>>> 2 * 7.5
15.0
>>> 15 / 3
5.0
```

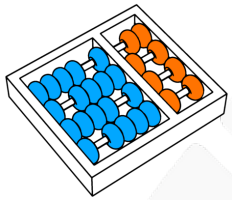


Operações aritméticas básicas

Algumas operações que podemos fazer:

- Soma: $+$.
- Subtração: $-$.
- Multiplicação: $*$.
- Divisão: $/$.
- Exponenciação: $**$.

```
>>> 3.2 + 7
10.2
>>> 5.3 - -2
7.3
>>> 2 * 7.5
15.0
>>> 15 / 3
5.0
>>> 2.5 ** 3
```

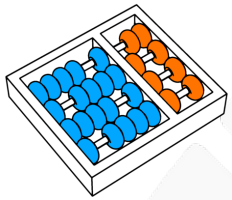


Operações aritméticas básicas

Algumas operações que podemos fazer:

- Soma: $+$.
- Subtração: $-$.
- Multiplicação: $*$.
- Divisão: $/$.
- Exponenciação: $**$.

```
>>> 3.2 + 7
10.2
>>> 5.3 - -2
7.3
>>> 2 * 7.5
15.0
>>> 15 / 3
5.0
>>> 2.5 ** 3
15.625
```

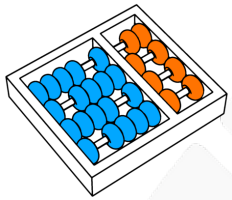


Operações aritméticas básicas

Algumas operações que podemos fazer:

- Soma: $+$.
- Subtração: $-$.
- Multiplicação: $*$.
- Divisão: $/$.
- Exponenciação: $**$.

```
>>> 3.2 + 7
10.2
>>> 5.3 - -2
7.3
>>> 2 * 7.5
15.0
>>> 15 / 3
5.0
>>> 2.5 ** 3
15.625
>>> 3 / 0
```



Operações aritméticas básicas

Algumas operações que podemos fazer:

- Soma: $+$.
- Subtração: $-$.
- Multiplicação: $*$.
- Divisão: $/$.
- Exponenciação: $**$.

```
>>> 3.2 + 7
10.2
>>> 5.3 - -2
7.3
>>> 2 * 7.5
15.0
>>> 15 / 3
5.0
>>> 2.5 ** 3
15.625
>>> 3 / 0
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
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

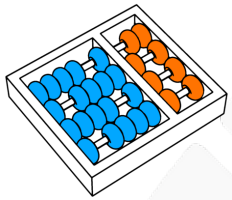
Temos a seguinte ordem:

- 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

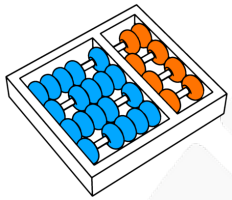
Temos a seguinte ordem:

- 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.
  - Chaves e colchetes não!

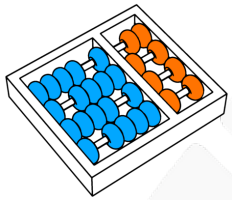


Precedência de operadores

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.
  - Chaves e colchetes não!

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

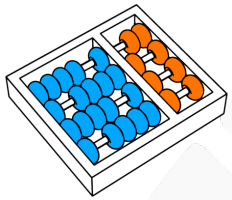


Precedência de operadores

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.
  - Chaves e colchetes não!

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

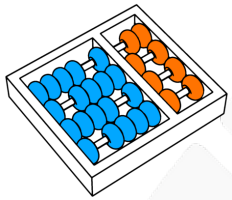


Precedência de operadores

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.
  - Chaves e colchetes não!

```
>>> 2 + 3 * 7
23
```

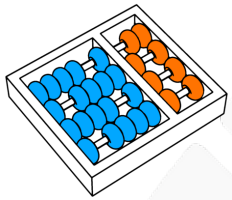


Precedência de operadores

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.
  - Chaves e colchetes não!

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

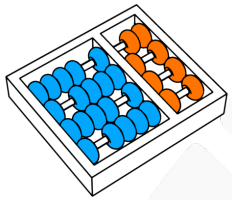


Precedência de operadores

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.
  - Chaves e colchetes não!

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

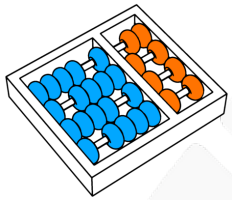


Precedência de operadores

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.
  - Chaves e colchetes não!

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

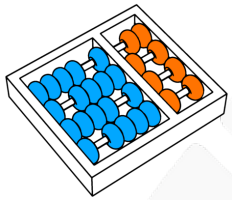


Precedência de operadores

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.
  - Chaves e colchetes não!

```
>>> 2 + 3 * 7
23
>>> (2 + 3) * 7
35
>>> ((2 + 3) * 7 + 15) * 2
100
```

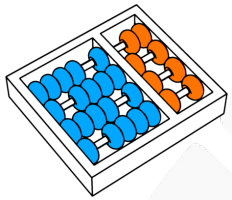


Precedência de operadores

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.
  - Chaves e colchetes não!

```
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.
  - Chaves e colchetes não!

```
>>> 2 + 3 * 7
23
>>> (2 + 3) * 7
35
>>> ((2 + 3) * 7 + 15) * 2
100
>>> [(2 + 3) * 7 + 15] * 2
[50, 50]
```

Movimento Uniformemente Variado

Calcular a posição final **s** de um objeto que:

Movimento Uniformemente Variado

Calcular a posição final s de um objeto que:

- Inicialmente está na posição s_0.

Movimento Uniformemente Variado

Calcular a posição final s de um objeto que:

- Inicialmente está na posição s_0.
- Se move:

Movimento Uniformemente Variado

Calcular a posição final s de um objeto que:

- Inicialmente está na posição s_0.
- Se move:
  - Durante t segundos.

Movimento Uniformemente Variado

Calcular a posição final s de um objeto que:

- Inicialmente está na posição s_0.
- Se move:
  - Durante t segundos.
  - Com aceleração constante a.

Movimento Uniformemente Variado

Calcular a posição final s de um objeto que:

- Inicialmente está na posição s_0.
- Se move:
  - Durante t segundos.
  - Com aceleração constante a.
  - Com velocidade inicial v_0.

Movimento Uniformemente Variado

Calcular a posição final s de um objeto que:

- Inicialmente está na posição s_0.
- Se move:
  - Durante t segundos.
  - Com aceleração constante a.
  - 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 que:

- Inicialmente está na posição s_0.
- Se move:
  - Durante t segundos.
  - Com aceleração constante a.
  - Com velocidade inicial v_0.

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

Em Python, algo do tipo:

Movimento Uniformemente Variado

Calcular a posição final s de um objeto que:

- Inicialmente está na posição s_0.
- Se move:
  - Durante t segundos.
  - Com aceleração constante a.
  - 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
```

Movimento Uniformemente Variado

Calcular a posição final s de um objeto que:

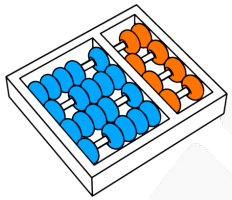
- Inicialmente está na posição s_0.
- Se move:
  - Durante t segundos.
  - Com aceleração constante a.
  - Com velocidade inicial v_0.

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

Em Python, algo do tipo:

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

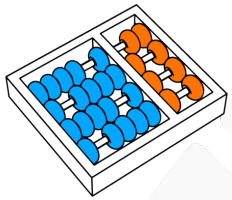
E se pudéssemos guardar os valores de s_0, a, t e v_0?



Armazenando informação

Algo mais longo, mas mais legível:

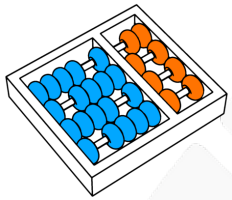
```
1 >>> s0 = 10
```



Armazenando informação

Algo mais longo, mas mais legível:

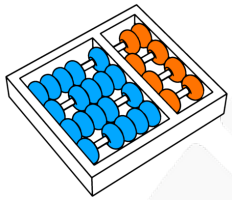
```
1 >>> s0 = 10
```



Armazenando informação

Algo mais longo, mas mais legível:

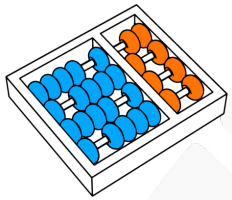
```
>>> s0 = 10
>>> v0 = 2
>>> t = 8
>>> a = 3
```



Armazenando informação

Algo mais longo, mas mais legível:

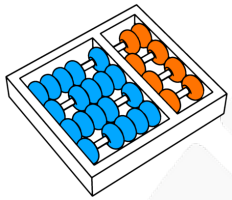
```
>>> s0 = 10
>>> v0 = 2
>>> t = 8
>>> a = 3
>>> s = s0 + v0 * t + a * t ** 2 / 2
```



Armazenando informação

Algo mais longo, mas mais legível:

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

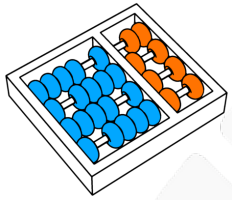


Armazenando informação

Algo mais longo, mas mais legível:

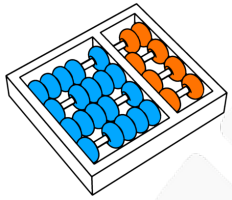
```
>>> s0 = 10
>>> v0 = 2
>>> t = 8
>>> a = 3
>>> s = s0 + v0 * t + a * t ** 2 / 2
>>> s
122.0
```

Variáveis e constantes



Variáveis

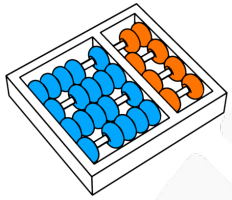
Variáveis são nomes para regiões da memória do computador:



Variáveis

Variáveis são nomes para regiões da memória do computador:

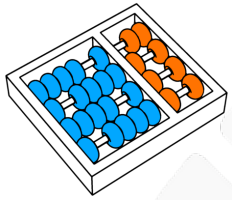
- Servem para armazenar informações.



Variáveis

Variáveis são nomes para regiões da memória do computador:

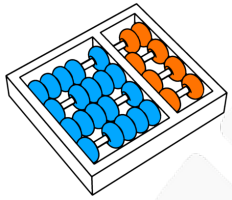
- Servem para armazenar informações.
- Com um nome bem definido.



Variáveis

Variáveis são nomes para regiões da memória do computador:

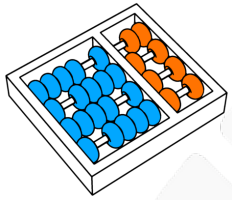
- Servem para armazenar informações.
- Com um nome bem definido.
- Mais fácil de lembrar do que um endereço da memória.



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.

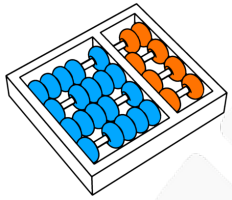


Variáveis

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!



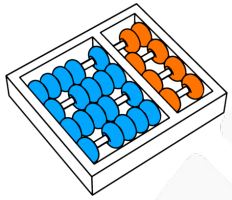
Variáveis

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
```



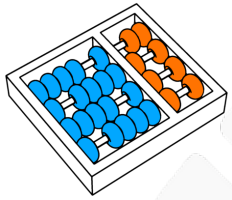
Variáveis

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
```



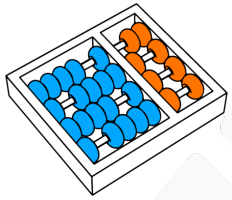
Variáveis

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
```



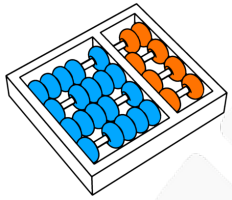
Variáveis

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
```



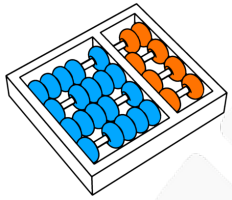
Variáveis

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
```



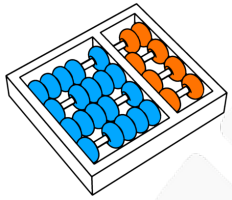
Variáveis

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

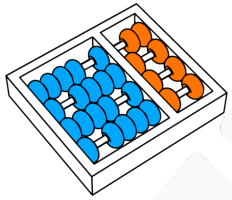
- Precisa começar com uma letra ou _ (underscore).

Regras para nomes de variáveis

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

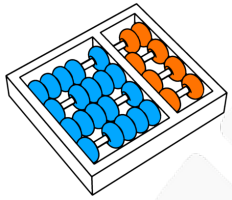
Regras para nomes de variáveis

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



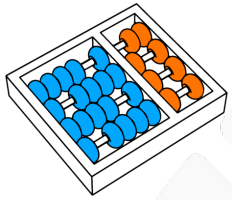
Regras para nomes de variáveis

- 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 _:



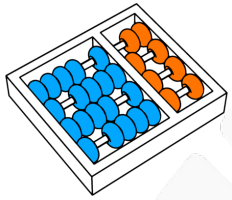
Regras para nomes de variáveis

- 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**.



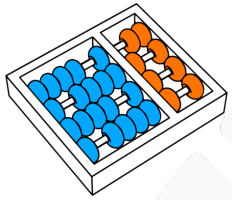
Regras para nomes de variáveis

- 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**.



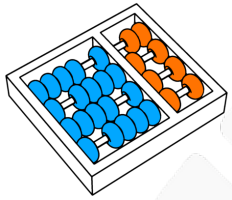
Regras para nomes de variáveis

- 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:



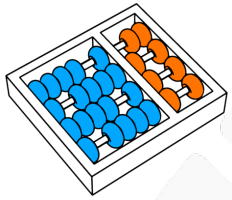
Regras para nomes de variáveis

- 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:

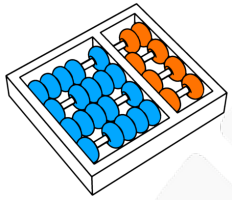


Regras para nomes de variáveis

- 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**, entre outros).

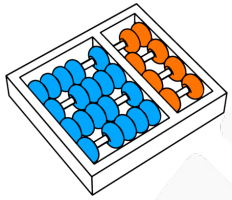
Bons nomes de variável

- Use nomes significativos:



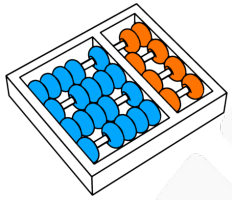
Bons nomes de variável

- Use nomes significativos:
  - **nome** é melhor do que **n**.



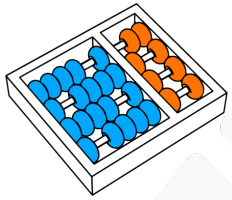
Bons nomes de variável

- Use nomes significativos:
  - **nome** é melhor do que **n**.
- Mas evite nomes muito grandes:



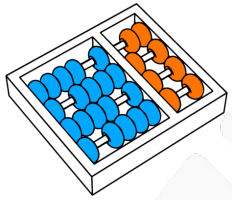
Bons nomes de variável

- Use nomes significativos:
  - **nome** é melhor do que **n**.
- Mas evite nomes muito grandes:
  - **nome** é melhor do que **nome_do_aluno_de_MC102**.



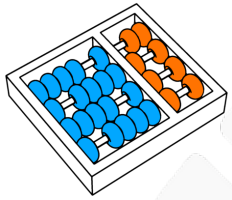
Bons nomes de variável

- 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:



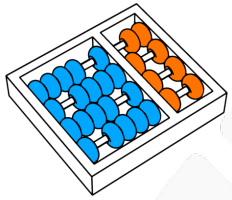
Bons nomes de variável

- 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**?



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

Constantes são os valores que não mudam:

Variáveis, constantes e atribuição

Constantes são os valores que não mudam:

- Ex: 2.34, -7.107, 10.

Variáveis, constantes e atribuição

Constantes são os valores que não mudam:

- Ex: 2.34, -7.107, 10.

Atribuição é a operação de armazenar um valor em uma variável:

Variáveis, constantes e atribuição

Constantes são os valores que não mudam:

- Ex: 2.34, -7.107, 10.

Atribuição é a operação de armazenar um valor em uma variável:

- O operador é o $=$.

Variáveis, constantes e atribuição

Constantes são os valores que não mudam:

- Ex: 2.34, -7.107, 10.

Atribuição é a operação de armazenar um valor em uma variável:

- O operador é o $=$.
- Podemos armazenar uma constante:

Variáveis, constantes e atribuição

Constantes são os valores que não mudam:

- Ex: 2.34, -7.107, 10.

Atribuição é a operação de armazenar um valor em uma variável:

- O operador é o $=$.
- Podemos armazenar uma constante:
  - Ex: $\mathbf{x} = 10$.

Variáveis, constantes e atribuição

Constantes são os valores que não mudam:

- Ex: 2.34, -7.107, 10.

Atribuição é a operação de armazenar um valor em uma variável:

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

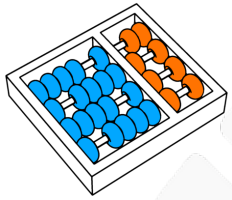
Variáveis, constantes e atribuição

Constantes são os valores que não mudam:

- Ex: 2.34, -7.107, 10.

Atribuição é a operação de armazenar um valor em uma variável:

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



Variáveis, constantes e atribuição

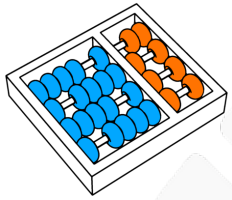
Constantes são os valores que não mudam:

- Ex: 2.34, -7.107, 10.

Atribuição é 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

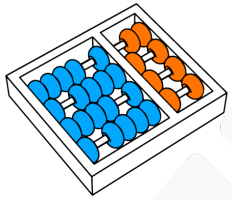
Constantes são os valores que não mudam:

- Ex: 2.34, -7.107, 10.

Atribuição é 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

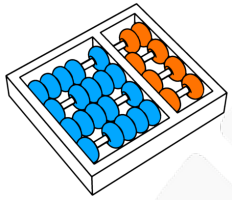
Constantes são os valores que não mudam:

- Ex: 2.34, -7.107, 10.

Atribuição é 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

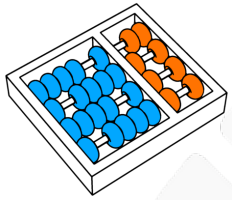
Constantes são os valores que não mudam:

- Ex: 2.34, -7.107, 10.

Atribuição é 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: $x = y$.

```
>>> y = 10
>>> y
10
>>> x = y
>>> x
10
>>> y = 15
>>> y
15
```



Variáveis, constantes e atribuição

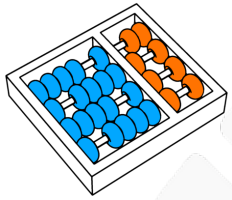
Constantes são os valores que não mudam:

- Ex: 2.34, -7.107, 10.

Atribuição é 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: $x = y$.

```
>>> y = 10
>>> y
10
>>> x = y
>>> x
10
>>> y = 15
>>> y
15
>>> x
10
```



Variáveis, constantes e atribuição

Constantes são os valores que não mudam:

- Ex: 2.34, -7.107, 10.

Atribuição é 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: $x = y$.

```
>>> y = 10
>>> y
10
>>> x = y
>>> x
10
>>> y = 15
>>> y
15
>>> x
10
```

Detalhes

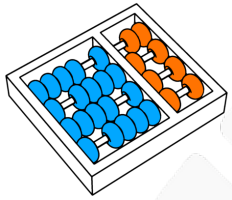
O que acontece nesse código?

```
1 >>> x
```

Detalhes

O que acontece nesse código?

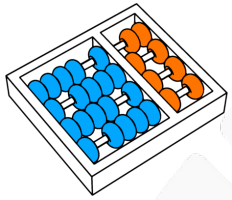
```
1 >>> x
```



Detalhes

O que acontece nesse código?

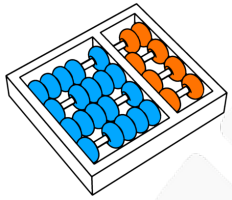
```
>>> x
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'x' is not defined
```



Detalhes

O que acontece nesse código?

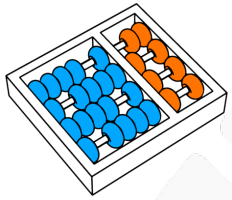
```
>>> x
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'x' is not defined
>>> x = 10
```



Detalhes

O que acontece nesse código?

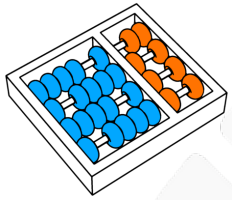
```
>>> x
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'x' is not defined
>>> x = 10
>>> x
```



Detalhes

O que acontece nesse código?

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

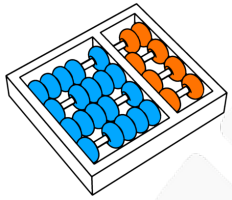


Detalhes

O que acontece nesse código?

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

A variável passa a existir quando recebe a primeira atribuição!



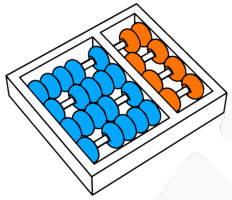
Detalhes

O que acontece nesse código?

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

A variável passa a existir quando recebe a primeira atribuição!

E nesse?



Detalhes

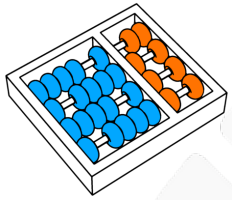
O que acontece nesse código?

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

A variável passa a existir quando recebe a primeira atribuição!

E nesse?

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



Detalhes

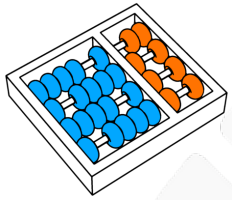
O que acontece nesse código?

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

A variável passa a existir quando recebe a primeira atribuição!

E nesse?

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



Detalhes

O que acontece nesse código?

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

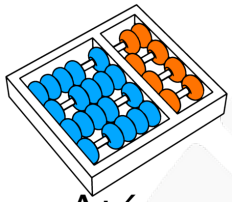
A variável passa a existir quando recebe a primeira atribuição!

E nesse?

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

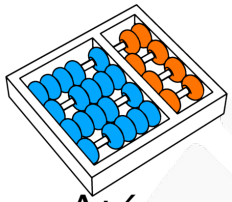
O = é uma atribuição de valor, não uma igualdade matemática!

Tipos de dados



Além de números

Até o momento trabalhamos apenas com números. . .



Além de números

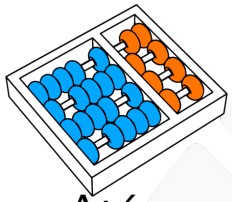
Até o momento trabalhamos apenas com números. . .

- Mas variáveis podem guardar muitos outros dados!

Além de números

Até o momento trabalhamos apenas com números. . .

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

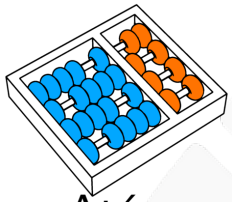


Além de números

Até o momento trabalhamos apenas com números. . .

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

O **tipo** de uma informação define:



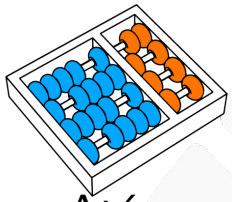
Além de números

Até o momento trabalhamos apenas com números. . .

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

O **tipo** de uma informação define:

- As operações que podemos fazer com ela.



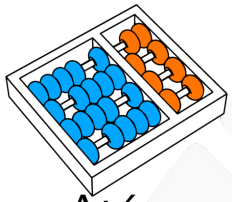
Além de números

Até o momento trabalhamos apenas com números. . .

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

O **tipo** de uma informação define:

- As operações que podemos fazer com ela.
- Qual é o resultado.



Além de números

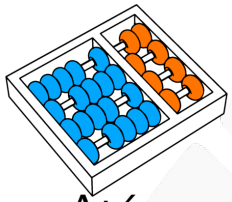
Até o momento trabalhamos apenas com números. . .

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

O **tipo** de uma informação define:

- As operações que podemos fazer com ela.
- Qual é o resultado.

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



Além de números

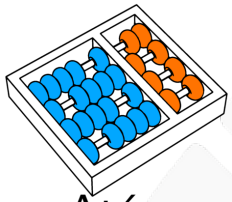
Até o momento trabalhamos apenas com números. . .

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

O **tipo** de uma informação define:

- As operações que podemos fazer com ela.
- Qual é o resultado.

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



Além de números

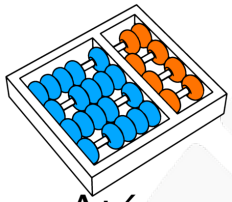
Até o momento trabalhamos apenas com números. . .

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

O **tipo** de uma informação define:

- As operações que podemos fazer com ela.
- Qual é o resultado.

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



Além de números

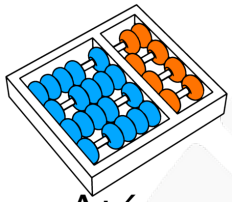
Até o momento trabalhamos apenas com números...

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

O **tipo** de uma informação define:

- As operações que podemos fazer com ela.
- Qual é o resultado.

```
>>> 10.3 + 5.2
15.5
>>> 'mc' + '102'
```



Além de números

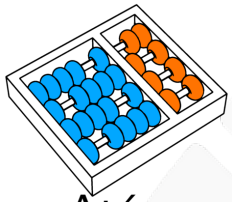
Até o momento trabalhamos apenas com números...

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

O **tipo** de uma informação define:

- As operações que podemos fazer com ela.
- Qual é o resultado.

```
>>> 10.3 + 5.2
15.5
>>> 'mc' + '102'
'mc102'
```



Além de números

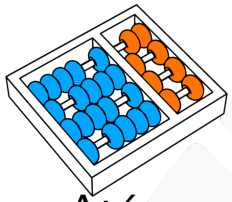
Até o momento trabalhamos apenas com números...

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

O **tipo** de uma informação define:

- As operações que podemos fazer com ela.
- Qual é o resultado.

```
>>> 10.3 + 5.2
15.5
>>> 'mc' + '102'
'mc102'
>>> 'mc' + 102
```



Além de números

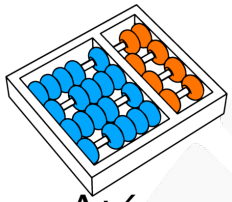
Até o momento trabalhamos apenas com números...

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

O **tipo** de uma informação define:

- As operações que podemos fazer com ela.
- Qual é o resultado.

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



Além de números

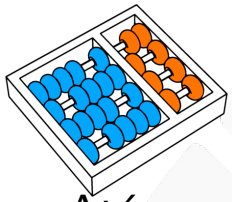
Até o momento trabalhamos apenas com números...

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

O **tipo** de uma informação define:

- As operações que podemos fazer com ela.
- Qual é o resultado.

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



Além de números

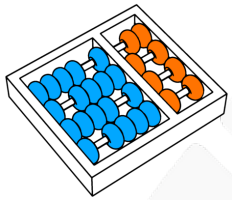
Até o momento trabalhamos apenas com números. . .

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

O **tipo** de uma informação define:

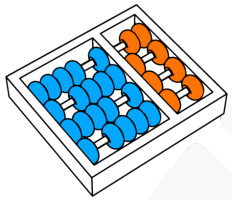
- As operações que podemos fazer com ela.
- Qual é o resultado.

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



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

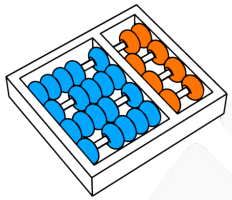
O tipo usado para representar números reais é o **float**:



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

O tipo usado para representar números reais é o **float**:

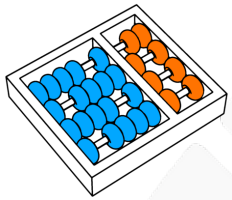
- Possui uma quantidade limitada de dígitos.



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

O tipo usado para representar números reais é o **float**:

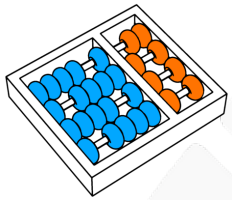
- Possui uma quantidade limitada de dígitos.
- Não pode representar qualquer número real:



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

O tipo usado para representar números reais é o **float**:

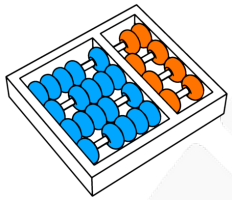
- Possui uma quantidade limitada de dígitos.
- Não pode representar qualquer número real:
  - π tem infinitos dígitos após a vírgula.



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

O tipo usado para representar números reais é o **float**:

- Possui uma quantidade limitada de dígitos.
- Não pode representar qualquer número real:
  - π tem infinitos dígitos após a vírgula.
- As operações podem ter erros de precisão.

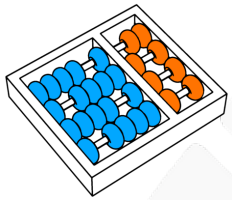


Números “reais” (ponto flutuante) - **float**

O tipo usado para representar números reais é o **float**:

- Possui uma quantidade limitada de dígitos.
- Não pode representar qualquer número real:
  - π tem infinitos dígitos após a vírgula.
- As operações podem ter erros de precisão.

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

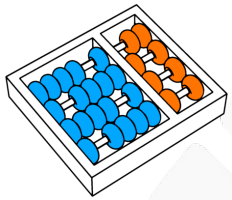


Números “reais” (ponto flutuante) - **float**

O tipo usado para representar números reais é o **float**:

- Possui uma quantidade limitada de dígitos.
- Não pode representar qualquer número real:
  - π tem infinitos dígitos após a vírgula.
- As operações podem ter erros de precisão.

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

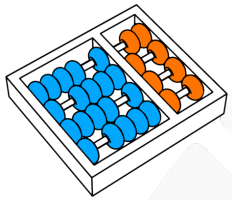


Números “reais” (ponto flutuante) - **float**

O tipo usado para representar números reais é o **float**:

- Possui uma quantidade limitada de dígitos.
- Não pode representar qualquer número real:
  - π tem infinitos dígitos após a vírgula.
- As operações podem ter erros de precisão.

```
>>> 15.3 / 3
5.1000000000000005
```

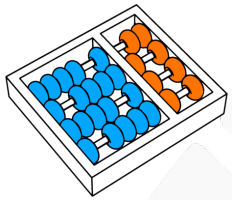


Números “reais” (ponto flutuante) - **float**

O tipo usado para representar números reais é o **float**:

- Possui uma quantidade limitada de dígitos.
- Não pode representar qualquer número real:
  - π tem infinitos dígitos após a vírgula.
- As operações podem ter erros de precisão.

```
>>> 15.3 / 3
5.1000000000000005
>>> 5.3 - 2.1
```

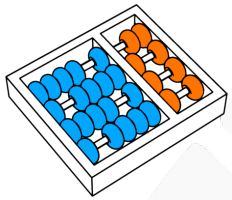


Números “reais” (ponto flutuante) - **float**

O tipo usado para representar números reais é o **float**:

- Possui uma quantidade limitada de dígitos.
- Não pode representar qualquer número real:
  - π tem infinitos dígitos após a vírgula.
- As operações podem ter erros de precisão.

```
>>> 15.3 / 3
5.1000000000000005
>>> 5.3 - 2.1
3.1999999999999997
```

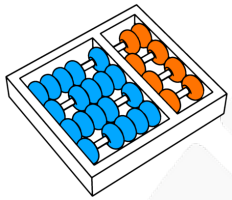


Números “reais” (ponto flutuante) - **float**

O tipo usado para representar números reais é o **float**:

- Possui uma quantidade limitada de dígitos.
- Não pode representar qualquer número real:
  - π tem infinitos dígitos após a vírgula.
- As operações podem ter erros de precisão.

```
>>> 15.3 / 3
5.1000000000000005
>>> 5.3 - 2.1
3.1999999999999997
>>> 5.3 + 1.9
```

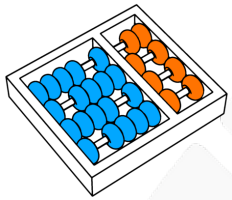


Números “reais” (ponto flutuante) - **float**

O tipo usado para representar números reais é o **float**:

- Possui uma quantidade limitada de dígitos.
- Não pode representar qualquer número real:
  - π tem infinitos dígitos após a vírgula.
- As operações podem ter erros de precisão.

```
>>> 15.3 / 3
5.1000000000000005
>>> 5.3 - 2.1
3.1999999999999997
>>> 5.3 + 1.9
7.199999999999999
```

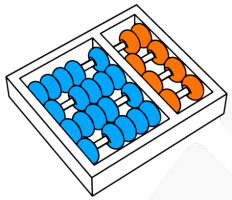


Números “reais” (ponto flutuante) - **float**

O tipo usado para representar números reais é o **float**:

- Possui uma quantidade limitada de dígitos.
- Não pode representar qualquer número real:
  - π tem infinitos dígitos após a vírgula.
- As operações podem ter erros de precisão.

```
>>> 15.3 / 3
5.1000000000000005
>>> 5.3 - 2.1
3.1999999999999997
>>> 5.3 + 1.9
7.1999999999999999
>>> 0.1 + 0.2
```

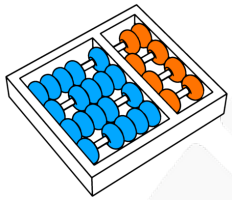


Números “reais” (ponto flutuante) - **float**

O tipo usado para representar números reais é o **float**:

- Possui uma quantidade limitada de dígitos.
- Não pode representar qualquer número real:
  - π tem infinitos dígitos após a vírgula.
- As operações podem ter erros de precisão.

```
>>> 15.3 / 3
5.1000000000000005
>>> 5.3 - 2.1
3.1999999999999997
>>> 5.3 + 1.9
7.1999999999999999
>>> 0.1 + 0.2
0.30000000000000004
```

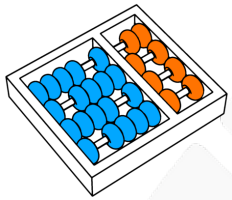


Números “reais” (ponto flutuante) - **float**

O tipo usado para representar números reais é o **float**:

- Possui uma quantidade limitada de dígitos.
- Não pode representar qualquer número real:
  - π tem infinitos dígitos após a vírgula.
- As operações podem ter erros de precisão.

```
>>> 15.3 / 3
5.1000000000000005
>>> 5.3 - 2.1
3.1999999999999997
>>> 5.3 + 1.9
7.1999999999999999
>>> 0.1 + 0.2
0.30000000000000004
>>> (15.6 / 3.0) * 3.0
```



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

O tipo usado para representar números reais é o **float**:

- Possui uma quantidade limitada de dígitos.
- Não pode representar qualquer número real:
  - π tem infinitos dígitos após a vírgula.
- As operações podem ter erros de precisão.

```
>>> 15.3 / 3
5.1000000000000005
>>> 5.3 - 2.1
3.1999999999999997
>>> 5.3 + 1.9
7.1999999999999999
>>> 0.1 + 0.2
0.30000000000000004
>>> (15.6 / 3.0) * 3.0
15.600000000000001
```

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

As constantes são representadas por:

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

As constantes são representadas por:

- Notação decimal: **4.2**, **3.1**, **−7.8**.

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

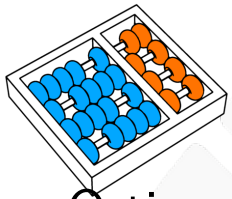
As constantes são representadas por:

- Notação decimal: **4.2**, **3.1**, **−7.8**.
  - Mesmo se for zero após o ponto, i.e., **2.0**.

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

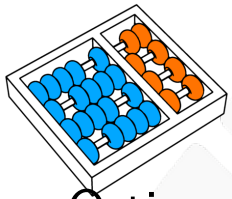
As constantes são representadas por:

- 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**

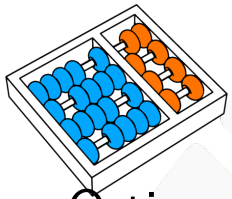
O tipo usado para representar números inteiros é o **int**:



Números inteiros - **int**

O tipo usado para representar números inteiros é o **int**:

- Com qualquer quantidade de dígitos!

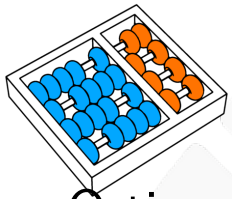


Números inteiros - **int**

O tipo usado para representar números inteiros é o **int**:

- Com qualquer quantidade de dígitos!

As constantes são representadas por:



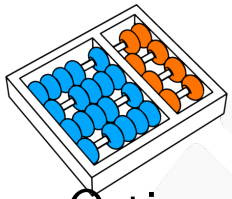
Números inteiros - **int**

O tipo usado para representar números inteiros é o **int**:

- Com qualquer quantidade de dígitos!

As constantes são representadas por:

- Números inteiros na base decimal (sem ponto):



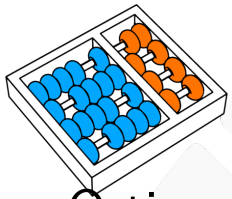
Números inteiros - **int**

O tipo usado para representar números inteiros é o **int**:

- Com qualquer quantidade de dígitos!

As constantes são representadas por:

- Números inteiros na base decimal (sem ponto):
  - Ex: $\mathbf{0}, \mathbf{2}, -\mathbf{3}$.



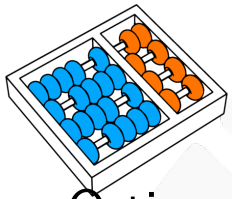
Números inteiros - **int**

O tipo usado para representar números inteiros é o **int**:

- Com qualquer quantidade de dígitos!

As constantes são representadas por:

- Números inteiros na base decimal (sem ponto):
  - Ex: $\mathbf{0}, \mathbf{2}, -\mathbf{3}$.
- Números inteiros na base binária precedidos por **0b**:



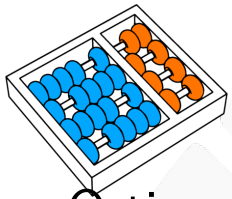
Números inteiros - **int**

O tipo usado para representar números inteiros é o **int**:

- Com qualquer quantidade de dígitos!

As constantes são representadas por:

- Números inteiros na base decimal (sem ponto):
  - Ex: $\mathbf{0}, \mathbf{2}, -\mathbf{3}$.
- Números inteiros na base binária precedidos por **0b**:
  - Ex: **0b11** é o número **3** em decimal.



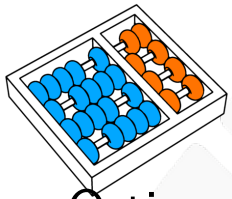
Números inteiros - **int**

O tipo usado para representar números inteiros é o **int**:

- Com qualquer quantidade de dígitos!

As constantes são representadas por:

- Números inteiros na base decimal (sem ponto):
  - Ex: $\mathbf{0}, \mathbf{2}, -\mathbf{3}$.
- Números inteiros na base binária precedidos por **0b**:
  - Ex: **0b11** é o número **3** em decimal.
- Números inteiros na base octal precedidos por **0o**:



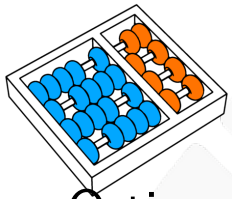
Números inteiros - **int**

O tipo usado para representar números inteiros é o **int**:

- Com qualquer quantidade de dígitos!

As constantes são representadas por:

- Números inteiros na base decimal (sem ponto):
  - Ex: $\mathbf{0}, \mathbf{2}, -\mathbf{3}$.
- 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.



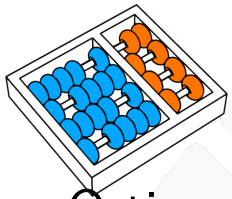
Números inteiros - **int**

O tipo usado para representar números inteiros é o **int**:

- Com qualquer quantidade de dígitos!

As constantes são representadas por:

- Números inteiros na base decimal (sem ponto):
  - Ex: $\mathbf{0}, \mathbf{2}, -\mathbf{3}$.
- 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**:



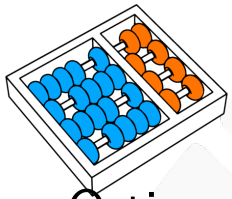
Números inteiros - **int**

O tipo usado para representar números inteiros é o **int**:

- Com qualquer quantidade de dígitos!

As constantes são representadas por:

- Números inteiros na base decimal (sem ponto):
  - Ex: $\mathbf{0}, \mathbf{2}, -\mathbf{3}$.
- 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**

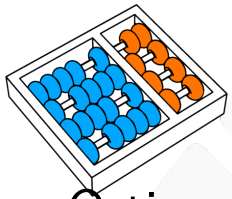
O tipo usado para representar números inteiros é o **int**:

- Com qualquer quantidade de dígitos!

As constantes são representadas por:

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

O tipo usado para representar números inteiros é o **int**:

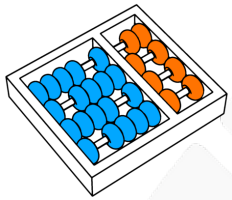
- Com qualquer quantidade de dígitos!

As constantes são representadas por:

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

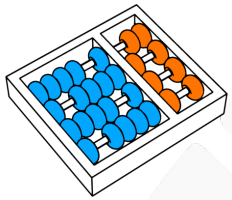
Essas três últimas são usadas em contextos mais específicos.

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



Dividindo a pizza

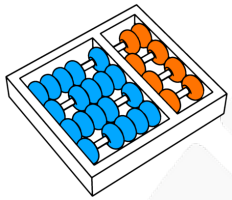
Queremos dividir **2** pizzas igualmente entre **5** alunos.



Dividindo a pizza

Queremos dividir **2** pizzas igualmente entre **5** alunos.

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

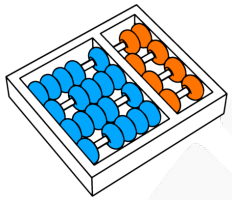


Dividindo a pizza

Queremos dividir **2** pizzas igualmente entre **5** alunos.

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

Vamos pedir ajuda para o Python!



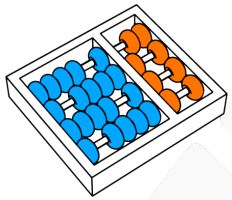
Dividindo a pizza

Queremos dividir **2** pizzas igualmente entre **5** alunos.

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

Vamos pedir ajuda para o Python!

```
>>> 16 / 5
3.2
```



Dividindo a pizza

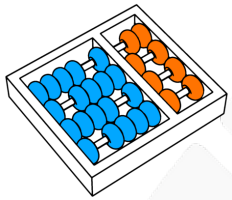
Queremos dividir **2** pizzas igualmente entre **5** alunos.

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

Vamos pedir ajuda para o Python!

```
>>> 16 / 5
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.

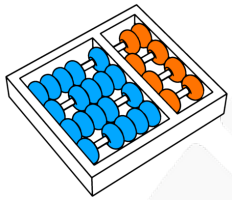
- Mas estamos com preguiça de fazer a conta...

Vamos pedir ajuda para o Python!

```
>>> 16 / 5
3.2
```

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

```
>>> 16 // 5
3
```



Dividindo a pizza

Queremos dividir 2 pizzas igualmente entre 5 alunos.

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

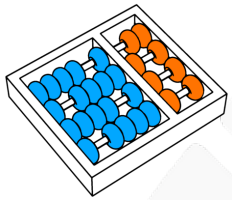
Vamos pedir ajuda para o Python!

```
>>> 16 / 5
3.2
```

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

```
>>> 16 // 5
3
```

O operador de divisão inteira é //.



Dividindo a pizza

Queremos dividir 2 pizzas igualmente entre 5 alunos.

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

Vamos pedir ajuda para o Python!

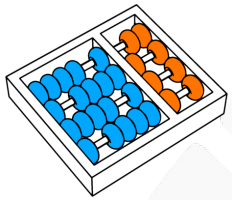
```
>>> 16 / 5
3.2
```

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

```
>>> 16 // 5
3
```

O operador de divisão inteira é //.

Quantas fatias sobram?



Dividindo a pizza

Queremos dividir 2 pizzas igualmente entre 5 alunos.

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

Vamos pedir ajuda para o Python!

```
>>> 16 / 5
3.2
```

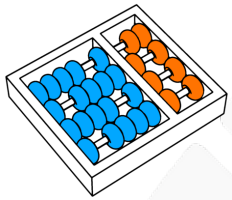
Não dá para cada um comer 3.2 pedaços... Vamos tentar de novo!

```
>>> 16 // 5
3
```

O operador de divisão inteira é //.

Quantas fatias sobram?

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



Dividindo a pizza

Queremos dividir 2 pizzas igualmente entre 5 alunos.

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

Vamos pedir ajuda para o Python!

```
>>> 16 / 5
3.2
```

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

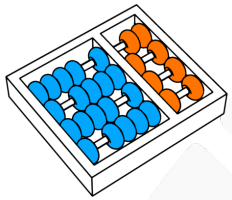
```
>>> 16 // 5
3
```

O operador de divisão inteira é //.

Quantas fatias sobram?

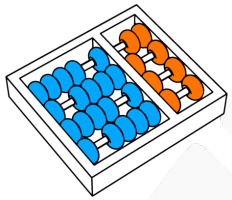
```
>>> 16 % 5
1
```

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



Tipo dos resultados

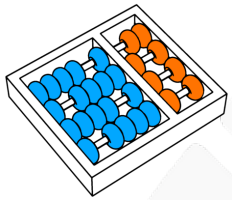
E se nossas expressões misturarem **float** e **int**?



Tipo dos resultados

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

```
>>> 2.0 + 3
5.0
```

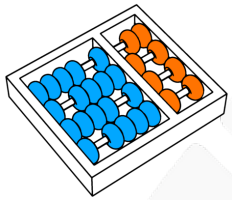


Tipo dos resultados

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

```
>>> 2.0 + 3
5.0
```

A expressão **x**/**y** sempre resulta em **float**.



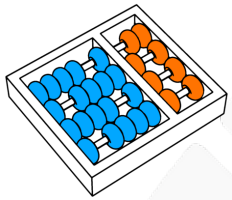
Tipo dos resultados

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

```
>>> 2.0 + 3
5.0
```

A expressão $\mathbf{x}/\mathbf{y}$ sempre resulta em **float**.

Expressões $\mathbf{x}+\mathbf{y}$, $\mathbf{x}-\mathbf{y}$, $\mathbf{x}*\mathbf{y}$, $\mathbf{x}**\mathbf{y}$, $\mathbf{x}//\mathbf{y}$ e $\mathbf{x}\%\mathbf{y}$:



Tipo dos resultados

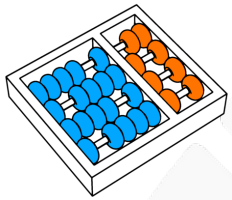
E se nossas expressões misturarem **float** e **int**?

```
>>> 2.0 + 3
5.0
```

A expressão $\mathbf{x}/\mathbf{y}$ sempre resulta em **float**.

Expressões $\mathbf{x}+\mathbf{y}$, $\mathbf{x}-\mathbf{y}$, $\mathbf{x}*\mathbf{y}$, $\mathbf{x}**\mathbf{y}$, $\mathbf{x}//\mathbf{y}$ e $\mathbf{x}\%\mathbf{y}$:

- Resultam em **int** se $\mathbf{x}$ e $\mathbf{y}$ são ambos **int**.



Tipo dos resultados

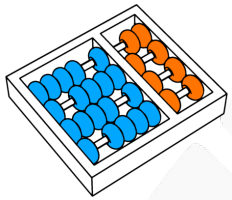
E se nossas expressões misturarem **float** e **int**?

```
>>> 2.0 + 3
5.0
```

A expressão $\mathbf{x}/\mathbf{y}$ sempre resulta em **float**.

Expressões $\mathbf{x}+\mathbf{y}$, $\mathbf{x}-\mathbf{y}$, $\mathbf{x}*\mathbf{y}$, $\mathbf{x}**\mathbf{y}$, $\mathbf{x}//\mathbf{y}$ e $\mathbf{x}\%\mathbf{y}$:

- Resultam em **int** se $\mathbf{x}$ e $\mathbf{y}$ são ambos **int**.
- Resultam em **float** se $\mathbf{x}$ ou $\mathbf{y}$ são **float**.



Tipo dos resultados

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

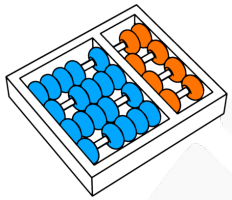
```
>>> 2.0 + 3
5.0
```

A expressão $\mathbf{x}/\mathbf{y}$ sempre resulta em **float**.

Expressões $\mathbf{x}+\mathbf{y}$, $\mathbf{x}-\mathbf{y}$, $\mathbf{x}*\mathbf{y}$, $\mathbf{x}**\mathbf{y}$, $\mathbf{x}//\mathbf{y}$ e $\mathbf{x}\%\mathbf{y}$:

- Resultam em **int** se $\mathbf{x}$ e $\mathbf{y}$ são ambos **int**.
- Resultam em **float** se $\mathbf{x}$ ou $\mathbf{y}$ são **float**.

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



Tipo dos resultados

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

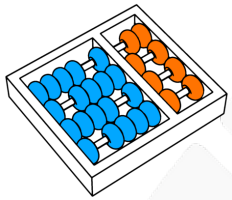
```
>>> 2.0 + 3
5.0
```

A expressão $\mathbf{x}/\mathbf{y}$ sempre resulta em **float**.

Expressões $\mathbf{x}+\mathbf{y}$, $\mathbf{x}-\mathbf{y}$, $\mathbf{x}*\mathbf{y}$, $\mathbf{x}**\mathbf{y}$, $\mathbf{x}//\mathbf{y}$ e $\mathbf{x}\%\mathbf{y}$:

- Resultam em **int** se $\mathbf{x}$ e $\mathbf{y}$ são ambos **int**.
- Resultam em **float** se $\mathbf{x}$ ou $\mathbf{y}$ são **float**.

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



Tipo dos resultados

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

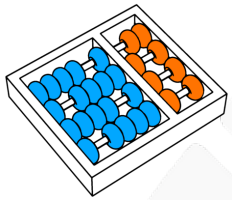
```
>>> 2.0 + 3
5.0
```

A expressão $\mathbf{x}/\mathbf{y}$ sempre resulta em **float**.

Expressões $\mathbf{x}+\mathbf{y}$, $\mathbf{x}-\mathbf{y}$, $\mathbf{x}*\mathbf{y}$, $\mathbf{x}**\mathbf{y}$, $\mathbf{x}//\mathbf{y}$ e $\mathbf{x}\%\mathbf{y}$:

- Resultam em **int** se $\mathbf{x}$ e $\mathbf{y}$ são ambos **int**.
- Resultam em **float** se $\mathbf{x}$ ou $\mathbf{y}$ são **float**.

```
>>> 16 // 5
3
```



Tipo dos resultados

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

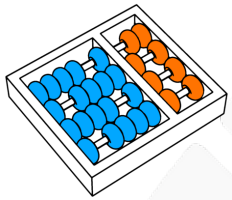
```
>>> 2.0 + 3
5.0
```

A expressão $\mathbf{x}/\mathbf{y}$ sempre resulta em **float**.

Expressões $\mathbf{x}+\mathbf{y}$, $\mathbf{x}-\mathbf{y}$, $\mathbf{x}*\mathbf{y}$, $\mathbf{x}**\mathbf{y}$, $\mathbf{x}//\mathbf{y}$ e $\mathbf{x}\%\mathbf{y}$:

- Resultam em **int** se $\mathbf{x}$ e $\mathbf{y}$ são ambos **int**.
- Resultam em **float** se $\mathbf{x}$ ou $\mathbf{y}$ são **float**.

```
>>> 16 // 5
3
>>> 16 // 5.0
```



Tipo dos resultados

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

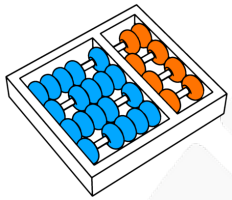
```
>>> 2.0 + 3
5.0
```

A expressão $\mathbf{x}/\mathbf{y}$ sempre resulta em **float**.

Expressões $\mathbf{x}+\mathbf{y}$, $\mathbf{x}-\mathbf{y}$, $\mathbf{x}*\mathbf{y}$, $\mathbf{x}**\mathbf{y}$, $\mathbf{x}//\mathbf{y}$ e $\mathbf{x}\%\mathbf{y}$:

- Resultam em **int** se $\mathbf{x}$ e $\mathbf{y}$ são ambos **int**.
- Resultam em **float** se $\mathbf{x}$ ou $\mathbf{y}$ são **float**.

```
>>> 16 // 5
3
>>> 16 // 5.0
3.0
```



Tipo dos resultados

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

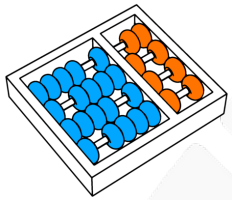
```
>>> 2.0 + 3
5.0
```

A expressão $\mathbf{x}/\mathbf{y}$ sempre resulta em **float**.

Expressões $\mathbf{x}+\mathbf{y}$, $\mathbf{x}-\mathbf{y}$, $\mathbf{x}*\mathbf{y}$, $\mathbf{x}**\mathbf{y}$, $\mathbf{x}//\mathbf{y}$ e $\mathbf{x}\%\mathbf{y}$:

- Resultam em **int** se $\mathbf{x}$ e $\mathbf{y}$ são ambos **int**.
- Resultam em **float** se $\mathbf{x}$ ou $\mathbf{y}$ são **float**.

```
>>> 16 // 5
3
>>> 16 // 5.0
3.0
>>> 16.0 % 5
```



Tipo dos resultados

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

```
>>> 2.0 + 3
5.0
```

A expressão $\mathbf{x}/\mathbf{y}$ sempre resulta em **float**.

Expressões $\mathbf{x}+\mathbf{y}$, $\mathbf{x}-\mathbf{y}$, $\mathbf{x}*\mathbf{y}$, $\mathbf{x}**\mathbf{y}$, $\mathbf{x}//\mathbf{y}$ e $\mathbf{x}\%\mathbf{y}$:

- Resultam em **int** se $\mathbf{x}$ e $\mathbf{y}$ são ambos **int**.
- Resultam em **float** se $\mathbf{x}$ ou $\mathbf{y}$ são **float**.

```
>>> 16 // 5
3
>>> 16 // 5.0
3.0
>>> 16.0 % 5
1.0
```

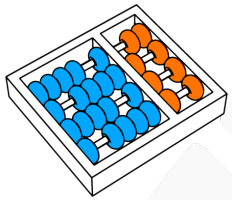
Textos - **str**

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 ':

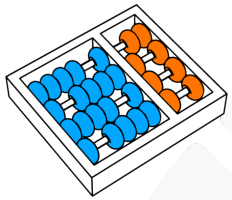
- Ex: "**uma string em Python**".

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'**.

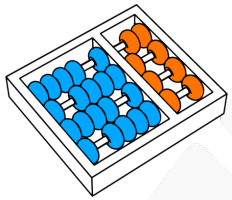


Textos - **str**

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.

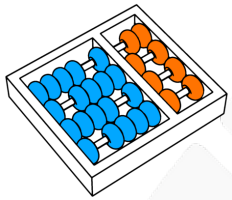


Textos - **str**

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.



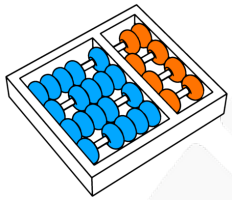
Textos - **str**

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**

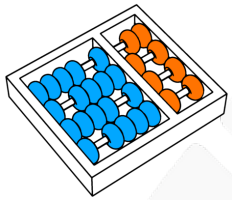
O tipo que representa texto no Python é o **str**.

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 """:

```
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 """:

```
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()**.

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

Verificando o tipo

Podemos descobrir o tipo usando **type()**.

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

Verificando o tipo

Podemos descobrir o tipo usando **type()**.

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

Verificando o tipo

Podemos descobrir o tipo usando **type()**.

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

Verificando o tipo

Podemos descobrir o tipo usando **type()**.

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

Verificando o tipo

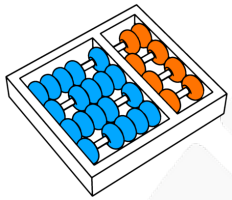
Podemos descobrir o tipo usando **type()**.

```
>>> type(2.0)
<class 'float'>
>>> type(2)
<class 'int'>
>>> type("2")
```

Verificando o tipo

Podemos descobrir o tipo usando **type()**.

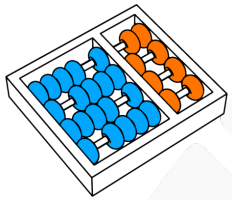
```
>>> type(2.0)
<class 'float'>
>>> type(2)
<class 'int'>
>>> type("2")
<class 'str'>
```



Verificando o tipo

Podemos descobrir o tipo usando **type()**.

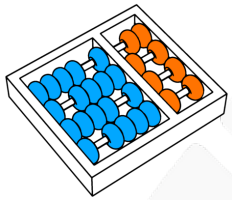
```
>>> type(2.0)
<class 'float'>
>>> type(2)
<class 'int'>
>>> type("2")
<class 'str'>
>>> x = 10 / 3
```



Verificando o tipo

Podemos descobrir o tipo usando **type()**.

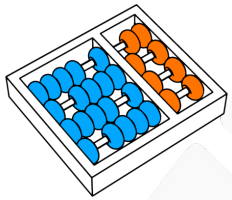
```
>>> type(2.0)
<class 'float'>
>>> type(2)
<class 'int'>
>>> type("2")
<class 'str'>
>>> x = 10 / 3
>>> type(x)
```



Verificando o tipo

Podemos descobrir o tipo usando **type()**.

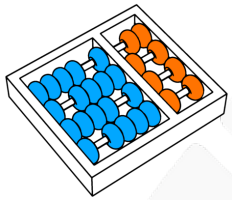
```
>>> type(2.0)
<class 'float'>
>>> type(2)
<class 'int'>
>>> type("2")
<class 'str'>
>>> x = 10 / 3
>>> type(x)
<class 'float'>
```



Verificando o tipo

Podemos descobrir o tipo usando **type()**.

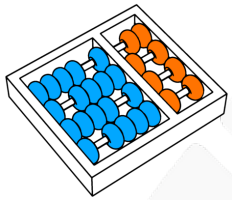
```
>>> type(2.0)
<class 'float'>
>>> type(2)
<class 'int'>
>>> type("2")
<class 'str'>
>>> x = 10 / 3
>>> type(x)
<class 'float'>
>>> y = 10 // 3
```



Verificando o tipo

Podemos descobrir o tipo usando **type()**.

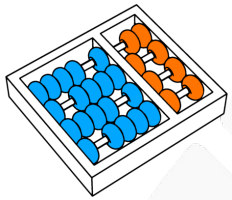
```
>>> type(2.0)
<class 'float'>
>>> type(2)
<class 'int'>
>>> type("2")
<class 'str'>
>>> x = 10 / 3
>>> type(x)
<class 'float'>
>>> y = 10 // 3
>>> type(y)
```



Verificando o tipo

Podemos descobrir o tipo usando **type()**.

```
>>> type(2.0)
<class 'float'>
>>> type(2)
<class 'int'>
>>> type("2")
<class 'str'>
>>> x = 10 / 3
>>> type(x)
<class 'float'>
>>> y = 10 // 3
>>> type(y)
<class 'int'>
```



Verificando o tipo

Podemos descobrir o tipo usando **type()**.

```
>>> type(2.0)
<class 'float'>
>>> type(2)
<class 'int'>
>>> type("2")
<class 'str'>
>>> x = 10 / 3
>>> type(x)
<class 'float'>
>>> y = 10 // 3
>>> type(y)
<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()**.

```
>>> str(3)
'3'
```

Convertendo entre tipos

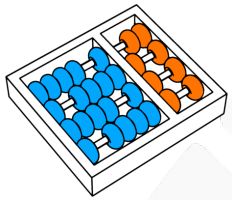
Podemos converter entre tipos usando **int()**, **float()** e **str()**.

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

Convertendo entre tipos

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

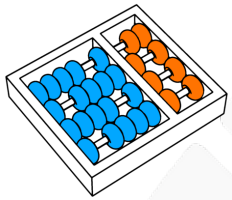
```
>>> str(3)
'3'
>>> str(3.5)
'3.5'
```



Convertendo entre tipos

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

```
>>> str(3)
'3'
>>> str(3.5)
'3.5'
>>> float("3.5")
```



Convertendo entre tipos

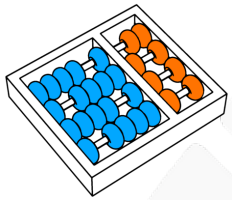
Podemos converter entre tipos usando **int()**, **float()** e **str()**.

```
>>> str(3)
'3'
>>> str(3.5)
'3.5'
>>> float("3.5")
3.5
```

Convertendo entre tipos

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

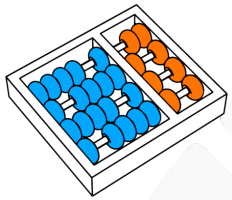
```
>>> str(3)
'3'
>>> str(3.5)
'3.5'
>>> float("3.5")
3.5
>>> float("3")
```



Convertendo entre tipos

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

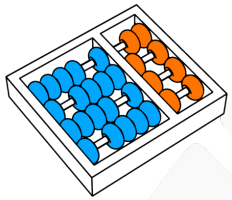
```
>>> str(3)
'3'
>>> str(3.5)
'3.5'
>>> float("3.5")
3.5
>>> float("3")
3.0
```



Convertendo entre tipos

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

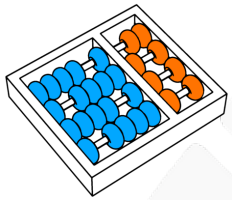
```
>>> str(3)
'3'
>>> str(3.5)
'3.5'
>>> float("3.5")
3.5
>>> float("3")
3.0
>>> int("3")
```



Convertendo entre tipos

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

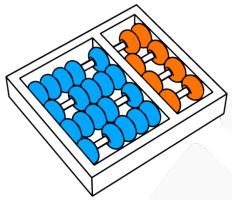
```
>>> str(3)
'3'
>>> str(3.5)
'3.5'
>>> float("3.5")
3.5
>>> float("3")
3.0
>>> int("3")
3
```



Convertendo entre tipos

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

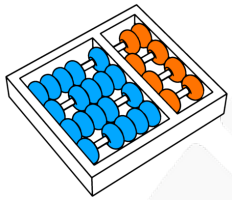
```
>>> str(3)
'3'
>>> str(3.5)
'3.5'
>>> float("3.5")
3.5
>>> float("3")
3.0
>>> int("3")
3
>>> int("3.5")
```



Convertendo entre tipos

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

```
>>> str(3)
'3'
>>> str(3.5)
'3.5'
>>> float("3.5")
3.5
>>> float("3")
3.0
>>> int("3")
3
>>> int("3.5")
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: invalid literal for int() with base 10: '3.5'
```



Convertendo entre tipos

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

```
>>> str(3)
'3'
>>> str(3.5)
'3.5'
>>> float("3.5")
3.5
>>> float("3")
3.0
>>> int("3")
3
>>> int("3.5")
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: invalid literal for int() with base 10: '3.5'
```

Veremos outras conversões de tipo durante o curso.

Armazenando o código

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:

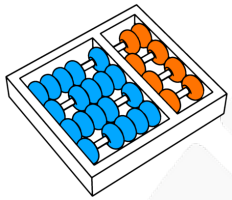
Exemplo de um arquivo .py

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

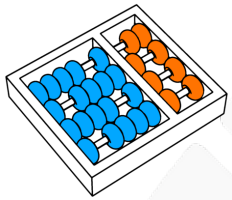
O que é impresso?

- Nada...

Impressão e leitura

Imprimindo dados

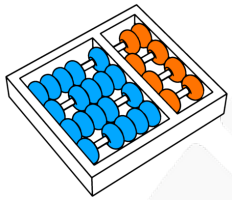
No Python, podemos usar **print** para imprimir informações.



Imprimindo dados

No Python, podemos usar **print** para imprimir informações.

```
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.

```
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 serem impressos:

Imprimindo dados

Podemos passar vários valores para serem impressos:

- Variáveis.

Imprimindo dados

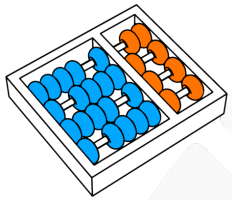
Podemos passar vários valores para serem impressos:

- Variáveis.
- Constantes.

Imprimindo dados

Podemos passar vários valores para serem impressos:

- Variáveis.
- Constantes.
- Até mesmo expressões.

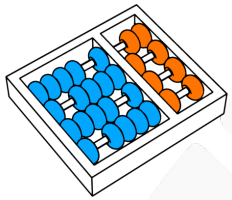


Imprimindo dados

Podemos passar vários valores para serem impressos:

- Variáveis.
- Constantes.
- Até mesmo expressões.

print(s0, v0, t, a) imprime: **10 2 8 3**



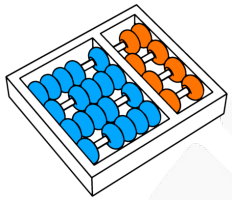
Imprimindo dados

Podemos passar vários valores para serem impressos:

- Variáveis.
- Constantes.
- 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

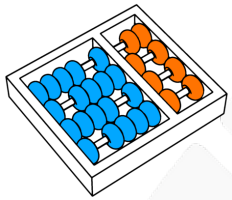
Podemos passar vários valores para serem impressos:

- Variáveis.
- Constantes.
- 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 serem impressos:

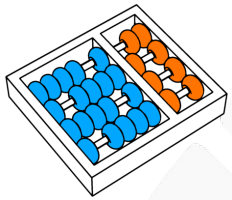
- Variáveis.
- Constantes.
- Até mesmo expressões.

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 serem impressos:

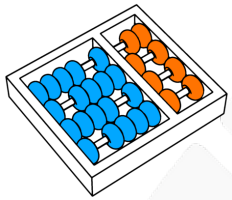
- Variáveis.
- Constantes.
- Até mesmo expressões.

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** e:



Imprimindo dados

Podemos passar vários valores para serem impressos:

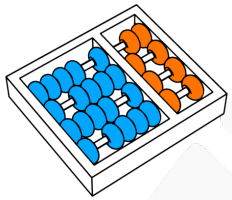
- Variáveis.
- Constantes.
- Até mesmo expressões.

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** e:
  - Não termina a linha.



Imprimindo dados

Podemos passar vários valores para serem impressos:

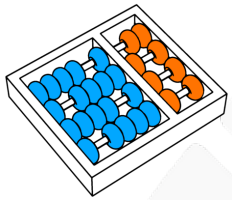
- Variáveis.
- Constantes.
- Até mesmo expressões.

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:

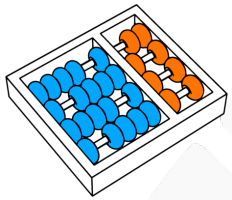
- **print(2, 3, sep=',')** imprime **2,3** (e termina a linha).
- **print(2, 3, sep=',', end='')** imprime **2,3** e:
  - Não termina a linha.
  - O próximo **print** começa na mesma linha.



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**.

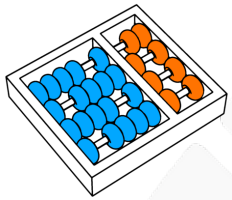
Ex: **x = input()**

Lendo dados

Podemos ler dados do teclado com **input**.

Ex: **x = input()**

input:



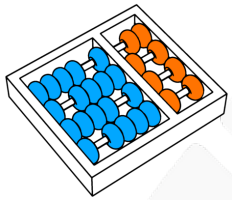
Lendo dados

Podemos ler dados do teclado com **input**.

Ex: **x = input()**

input:

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



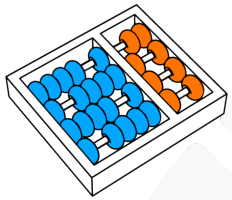
Lendo dados

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

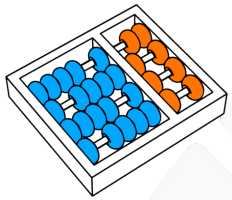
Podemos ler dados do teclado com **input**.

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

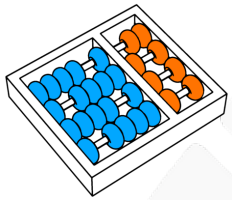
Podemos ler dados do teclado com **input**.

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**.

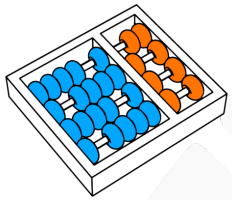
Ex: **x = input()**

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 quiseremos 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 quiseremos ler um **int** ou **float**?

- Usamos conversão de tipos!

Versão final

```
1 s0 = float(input('Entre com s0: '))
2 v0 = float(input('Etre 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)
```

Dicas

python prog1.py < entrada.txt > saida.txt

Dicas

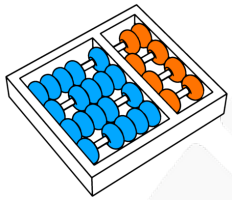
python prog1.py < entrada.txt > saida.txt

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

Dicas

python prog1.py < entrada.txt > saida.txt

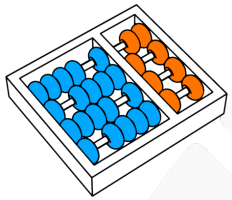
- < redireciona a entrada padrão para o arquivo **entrada.txt**.
- > redireciona a saída padrão para o arquivo **saida.txt**.



Dicas

python prog1.py < entrada.txt > saida.txt

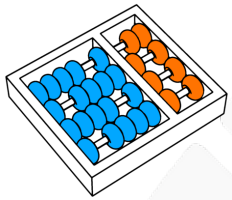
- < redireciona a entrada padrão para o arquivo **entrada.txt**.
- > redireciona a saída padrão para o arquivo **saida.txt**.
- Você pode usar apenas um dos dois se quiser.



Dicas

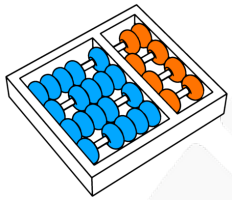
python prog1.py < entrada.txt > saida.txt

- < 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.
- Pode ser útil para os laboratórios!



Parede de zulejos

```
1 largura_azulejo = int(input('Entre a largura dos azulejos: '))
2 largura_parede = int(input('Etre a largura da parede: '))
3 altura_parede = int(input('Etre a altura da parede: '))
4 area_azulejo = largura_azulejo ** 2
5 area_parede = largura_parede * altura_parede
6 quantidade = (area_parede + area_azulejo - 1) // area_azulejo
7 print('Quantidade de azulejos requerida: ', quantidade)
```



Último dígito

```
1 numero = int(input('Entre o número: '))
2 digito = numero % 10
3 print('O último digito é: ', digito)
```

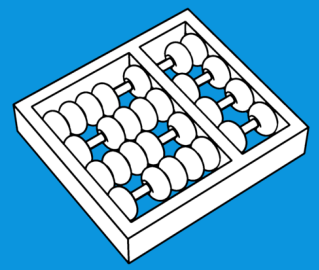
Exercício

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 juros é aplicado ao final do mês.

Python: primeiros passos

MC102 - Algoritmos e Programação de Computadores

Santiago Valdés Ravelo
https://ic.unicamp.br/~santiago/
ravelo@unicamp.br

UNICAMP