

PYTHON: PRIMEIROS PASSOS

MC102 - Algoritmos e
Programação de
Computadores

Santiago Valdés Ravelo
[https://ic.unicamp.br/~santiago/
ravelo@unicamp.br](https://ic.unicamp.br/~santiago/ravelo@unicamp.br)

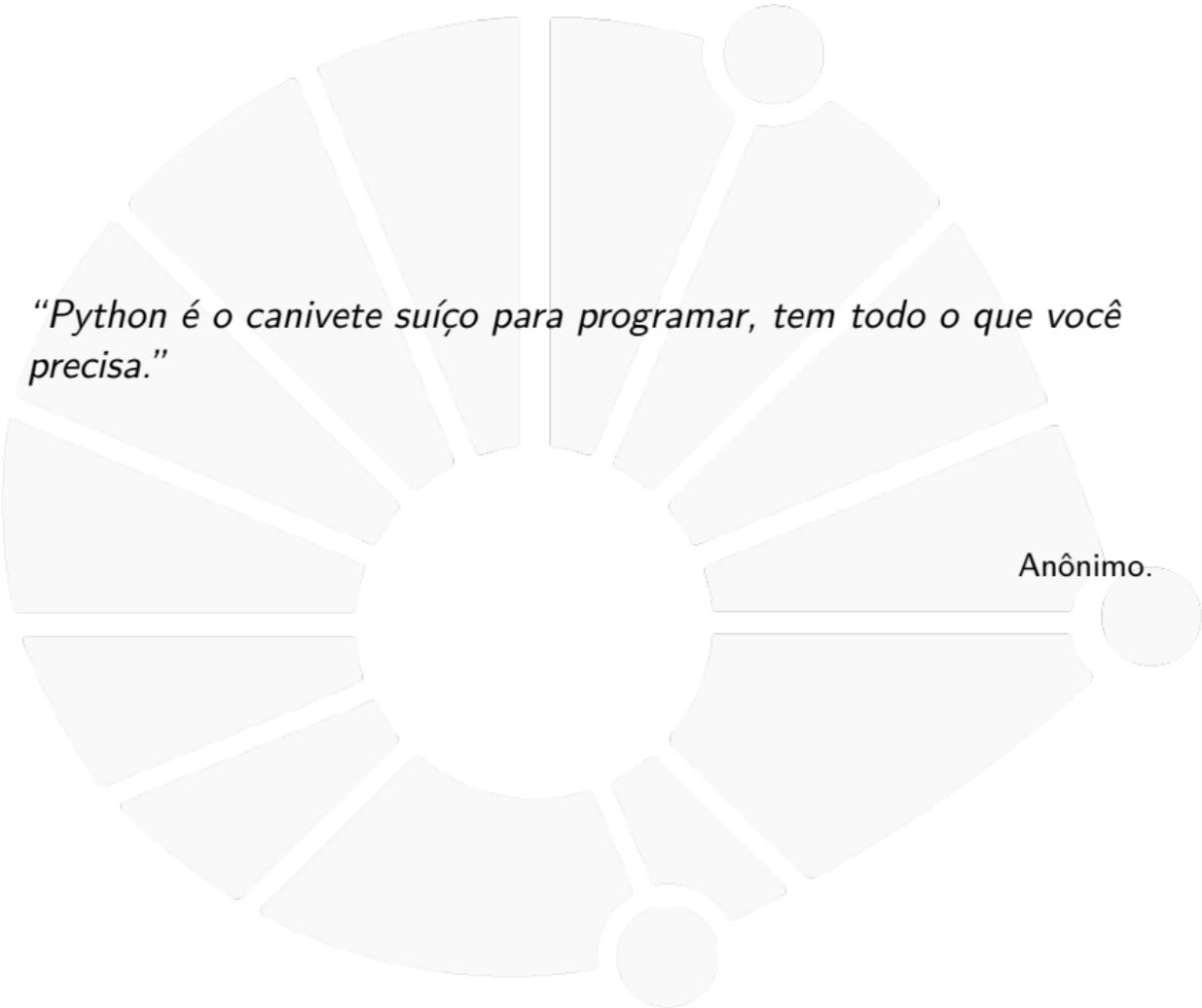
03/24

2



UNICAMP



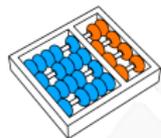


“Python é o canivete suíço para programar, tem todo o que você precisa.”

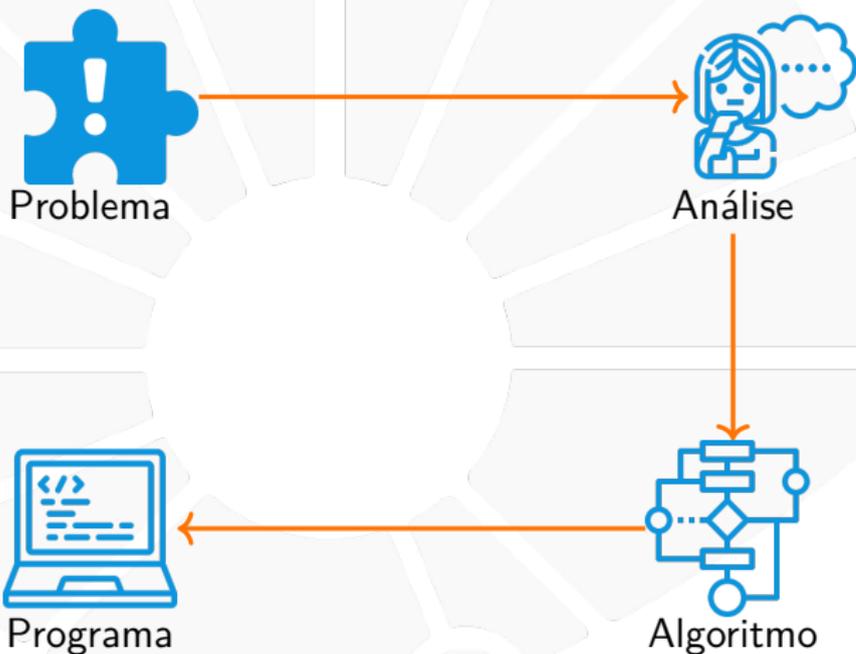
Anônimo.



CRIAÇÃO DE PROGRAMAS



Solução de um problema





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.

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*
-

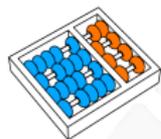


Algoritmo para um problema. Exemplo

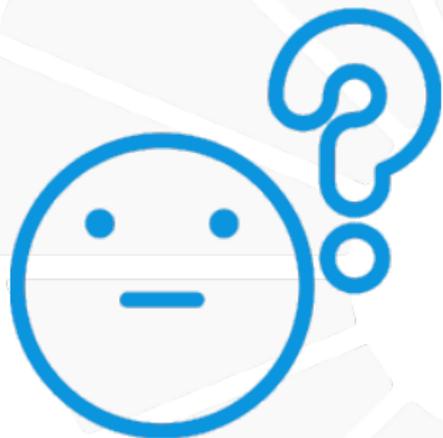
Algoritmo: ULTIMO-DIGITO

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

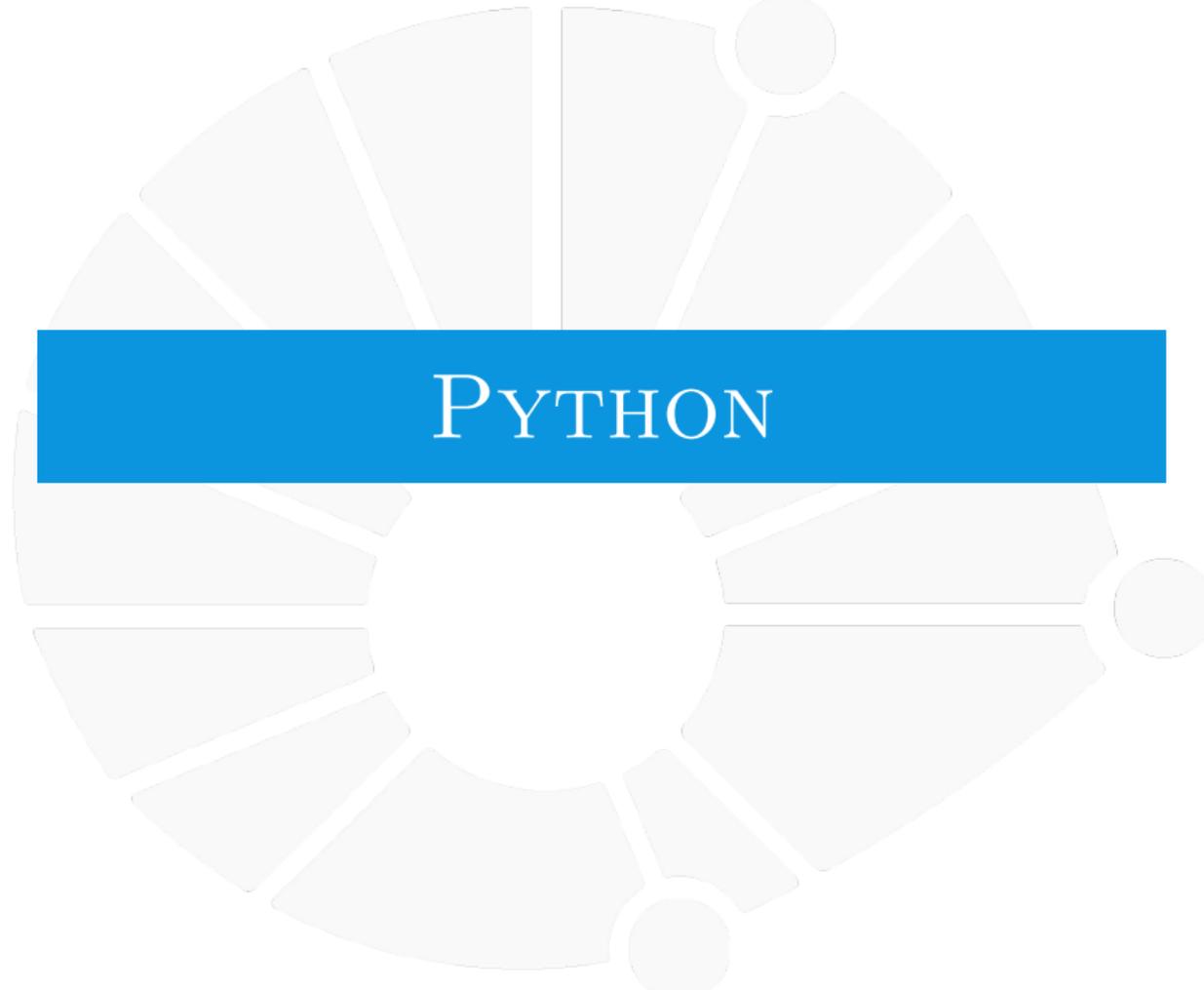
Criação de programas



Pergunta



Como programar um algoritmo?

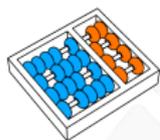
The image features a large, light gray watermark of the Python logo in the background. The logo consists of two interlocking snakes, one blue and one yellow, forming a circular shape. A solid blue horizontal bar is superimposed across the center of the logo.

PYTHON



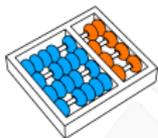
Características

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



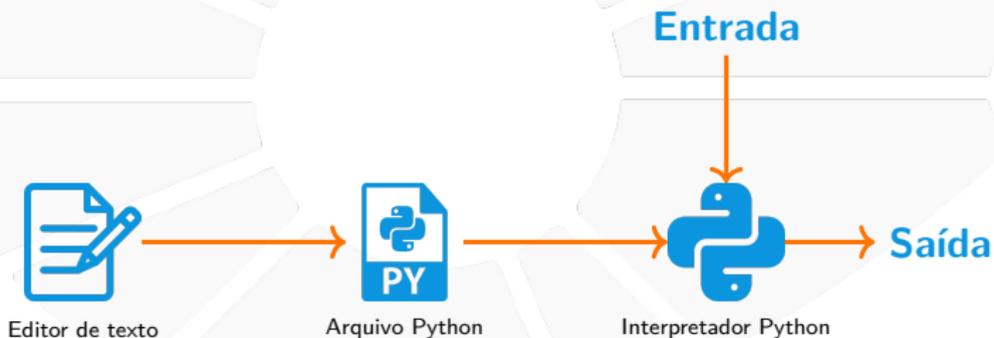
Onde encontrar?

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

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

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

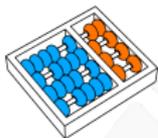


Precedência de operadores

Temos a seguinte ordem:

- ▶ 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
8 [50, 50]
```



Movimento Uniformemente Variado

Calcular a posição final s de um objeto que:

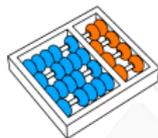
- ▶ Inicialmente está na posição s_0 .
- ▶ Se move:
 - ▶ Durante t segundos.
 - ▶ Com aceleração constante a .
 - ▶ Com velocidade inicial v_0 .

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

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 ?



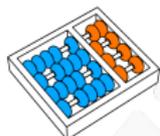
Armazenando informação

Algo mais longo, mas mais legível:

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



VARIÁVEIS E CONSTANTES



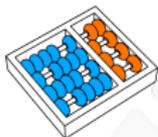
Variáveis

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.

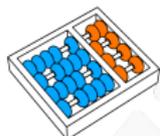
O nome “**variável**” vem do fato que o valor pode mudar!

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



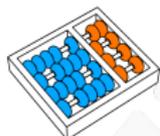
Regras para nomes de variáveis

- ▶ Precisa começar com uma letra ou `_` (underscore).
 - ▶ 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:
 - ▶ **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

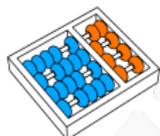
Constantes são os valores que não mudam:

- ▶ 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
7 >>> y = 15
8 >>> y
9 15
10 >>> x
11 10
```



Detalhes

O que acontece nesse código?

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

A variável passa a existir quando recebe a primeira atribuição!

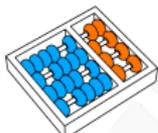
E nesse?

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

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



TIPOS DE DADOS



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

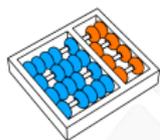


Números “reais” (ponto flutuante) - float

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
2 5.10000000000000005
3 >>> 5.3 - 2.1
4 3.1999999999999997
5 >>> 5.3 + 1.9
6 7.1999999999999999
7 >>> 0.1 + 0.2
8 0.30000000000000004
9 >>> (15.6 / 3.0) * 3.0
10 15.600000000000001
```



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`:

- ▶ 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.

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

Vamos pedir ajuda para o Python!

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

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

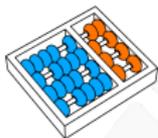
```
1 >>> 16 // 5
2 3
```

O operador de divisão inteira é `//`.

Quantas fatias sobram?

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

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



Tipo dos resultados

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

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

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

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

▶ Resultam em **int** se x e y são ambos **int**.

▶ Resultam em **float** se x ou y são **float**.

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



Textos - `str`

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

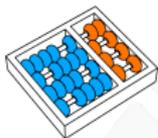
A constante é uma sequência de caracteres entre `"` ou `'`:

- ▶ 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."""
```

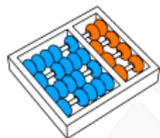
Veremos mais sobre `str` durante o curso!



Verificando o tipo

Podemos descobrir o tipo usando `type()`.

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



Convertendo entre tipos

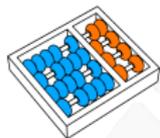
Podemos converter entre tipos usando `int()`, `float()` e `str()`.

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

Veremos outras conversões de tipo durante o curso.



ARMAZENANDO O CÓDIGO



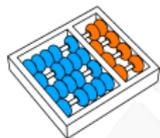
Arquivos .py

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):

```
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:

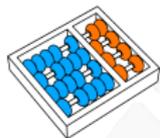
python prog1.py

O que é impresso?

▶ Nada...



IMPRESSÃO E LEITURA

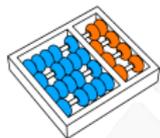


Imprimindo dados

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

```
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:

- ▶ 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)
```

Para calcular com outros valores é necessário alterar o arquivo!



Lendo dados

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

Ex: `x = input()`

input:

- ▶ Lê uma linha inteira de texto do terminal (até o enter).
- ▶ Devolve uma **string** com todo o texto.

Podemos também colocar uma mensagem para o usuário:

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

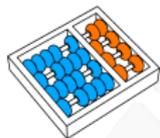
Mas e se quisermos ler um **int** ou **float**?

- ▶ Usamos conversão de tipos!



Versão final

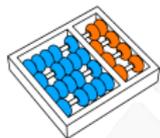
```
1 s0 = float(input('Entre com s0: '))
2 v0 = float(input('Entre com v0: '))
3 t = float(input('Entre com t: '))
4 a = float(input('Entre com a: '))
5 s = s0 + v0 * t + a * t ** 2 / 2
6 print('Posição final: ', s)
```



Dicas

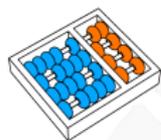
```
python prog1.py < entrada.txt > saida.txt
```

- ▶ < 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('Entre a largura da parede: '))
3 altura_parede = int(input('Entre 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 dígito 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](https://ic.unicamp.br/~santiago/ravelo@unicamp.br)

03/24

2



UNICAMP

