

# Curso de básico de programação em Python

## Encontro 1 - Apresentação da linguagem

Prof. Louis Augusto

`louis.augusto@ifsc.edu.br`



**INSTITUTO FEDERAL  
SANTA CATARINA**

Instituto Federal de Santa Catarina  
Campus São José

## 1 Python é a linguagem para aprender a programar

- Python é bem simples
- Python é uma linguagem não tipada
- Comentários em Python
- Codificação dos caracteres usados

## 2 Entrada e saída básica de dados

- Entrada de dados
- Saída de dados
- Método format
- Exercícios

## 3 Funções - parte 1

- Guardando códigos para reuso
- Retorno de funções
- Argumentos simples para funções

## 1 Python é a linguagem para aprender a programar

- Python é bem simples
- Python é uma linguagem não tipada
- Comentários em Python
- Codificação dos caracteres usados

## 2 Entrada e saída básica de dados

- Entrada de dados
- Saída de dados
- Método format
- Exercícios

## 3 Funções - parte 1

- Guardando códigos para reuso
- Retorno de funções
- Argumentos simples para funções

# Python é bem simples

Python foi uma linguagem criada nos anos 1990 para ser fácil de programar. É uma linguagem interpretada que roda por cima da linguagem C (precisa ter a linguagem C instalada na máquina).

Há vantagens e desvantagens em usar python:

- Não é necessário criar um arquivo objeto e um executável para poder rodar um programa. O Python utiliza o próprio código fonte para interpretar e executar os comandos.
- O tempo necessário para efetuar testes é mais rápido.
- A execução do programa é mais lenta comparada com linguagens compiladas, como C.
- Não é conveniente para construir programas que exijam uma estrutura de dados robusta.
- O tempo usado para construir um programa em Python para a maioria de tarefas simples é muito menor do que em outras linguagens compiladas.
- Tem uma vastíssima comunidade, que além de participar tirando dúvidas de outros usuários, constroem bibliotecas vastas, que podem ser instaladas em uma linha de comando.

# Python é bem simples

Python foi uma linguagem criada nos anos 1990 para ser fácil de programar. É uma linguagem interpretada que roda por cima da linguagem C (precisa ter a linguagem C instalada na máquina).

Há vantagens e desvantagens em usar python:

- Não é necessário criar um arquivo objeto e um executável para poder rodar um programa. O Python utiliza o próprio código fonte para interpretar e executar os comandos.
- O tempo necessário para efetuar testes é mais rápido.
- A execução do programa é mais lenta comparada com linguagens compiladas, como C.
- Não é conveniente para construir programas que exijam uma estrutura de dados robusta.
- O tempo usado para construir um programa em Python para a maioria de tarefas simples é muito menor do que em outras linguagens compiladas.
- Tem uma vastíssima comunidade, que além de participar tirando dúvidas de outros usuários, constroem bibliotecas vastas, que podem ser instaladas em uma linha de comando.

# Python é bem simples

Python foi uma linguagem criada nos anos 1990 para ser fácil de programar. É uma linguagem interpretada que roda por cima da linguagem C (precisa ter a linguagem C instalada na máquina).

Há vantagens e desvantagens em usar python:

- Não é necessário criar um arquivo objeto e um executável para poder rodar um programa. O Python utiliza o próprio código fonte para interpretar e executar os comandos.
- O tempo necessário para efetuar testes é mais rápido.
- A execução do programa é mais lenta comparada com linguagens compiladas, como C.
- Não é conveniente para construir programas que exijam uma estrutura de dados robusta.
- O tempo usado para construir um programa em Python para a maioria de tarefas simples é muito menor do que em outras linguagens compiladas.
- Tem uma vastíssima comunidade, que além de participar tirando dúvidas de outros usuários, constroem bibliotecas vastas, que podem ser instaladas em uma linha de comando.

# Python é bem simples

Python foi uma linguagem criada nos anos 1990 para ser fácil de programar. É uma linguagem interpretada que roda por cima da linguagem C (precisa ter a linguagem C instalada na máquina).

Há vantagens e desvantagens em usar python:

- Não é necessário criar um arquivo objeto e um executável para poder rodar um programa. O Python utiliza o próprio código fonte para interpretar e executar os comandos.
- O tempo necessário para efetuar testes é mais rápido.
- A execução do programa é mais lenta comparada com linguagens compiladas, como C.
- Não é conveniente para construir programas que exijam uma estrutura de dados robusta.
- O tempo usado para construir um programa em Python para a maioria de tarefas simples é muito menor do que em outras linguagens compiladas.
- Tem uma vastíssima comunidade, que além de participar tirando dúvidas de outros usuários, constroem bibliotecas vastas, que podem ser instaladas em uma linha de comando.

# Python é bem simples

Python foi uma linguagem criada nos anos 1990 para ser fácil de programar. É uma linguagem interpretada que roda por cima da linguagem C (precisa ter a linguagem C instalada na máquina).

Há vantagens e desvantagens em usar python:

- Não é necessário criar um arquivo objeto e um executável para poder rodar um programa. O Python utiliza o próprio código fonte para interpretar e executar os comandos.
- O tempo necessário para efetuar testes é mais rápido.
- A execução do programa é mais lenta comparada com linguagens compiladas, como C.
- Não é conveniente para construir programas que exijam uma estrutura de dados robusta.
- O tempo usado para construir um programa em Python para a maioria de tarefas simples é muito menor do que em outras linguagens compiladas.
- Tem uma vastíssima comunidade, que além de participar tirando dúvidas de outros usuários, constroem bibliotecas vastas, que podem ser instaladas em uma linha de comando.



# Python é bem simples

Python foi uma linguagem criada nos anos 1990 para ser fácil de programar. É uma linguagem interpretada que roda por cima da linguagem C (precisa ter a linguagem C instalada na máquina).

Há vantagens e desvantagens em usar python:

- Não é necessário criar um arquivo objeto e um executável para poder rodar um programa. O Python utiliza o próprio código fonte para interpretar e executar os comandos.
- O tempo necessário para efetuar testes é mais rápido.
- A execução do programa é mais lenta comparada com linguagens compiladas, como C.
- Não é conveniente para construir programas que exijam uma estrutura de dados robusta.
- O tempo usado para construir um programa em Python para a maioria de tarefas simples é muito menor do que em outras linguagens compiladas.
- Tem uma vastíssima comunidade, que além de participar tirando dúvidas de outros usuários, constroem bibliotecas vastas, que podem ser instaladas em uma linha de comando.

# Python é bem simples

Python foi uma linguagem criada nos anos 1990 para ser fácil de programar. É uma linguagem interpretada que roda por cima da linguagem C (precisa ter a linguagem C instalada na máquina).

Há vantagens e desvantagens em usar python:

- Não é necessário criar um arquivo objeto e um executável para poder rodar um programa. O Python utiliza o próprio código fonte para interpretar e executar os comandos.
- O tempo necessário para efetuar testes é mais rápido.
- A execução do programa é mais lenta comparada com linguagens compiladas, como C.
- Não é conveniente para construir programas que exijam uma estrutura de dados robusta.
- O tempo usado para construir um programa em Python para a maioria de tarefas simples é muito menor do que em outras linguagens compiladas.
- Tem uma vastíssima comunidade, que além de participar tirando dúvidas de outros usuários, constroem bibliotecas vastas, que podem ser instaladas em uma linha de comando.

# Python é bem simples

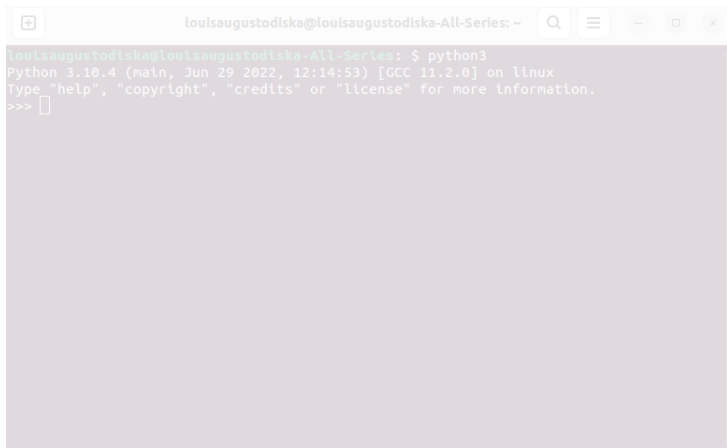
Python foi uma linguagem criada nos anos 1990 para ser fácil de programar. É uma linguagem interpretada que roda por cima da linguagem C (precisa ter a linguagem C instalada na máquina).

Há vantagens e desvantagens em usar python:

- Não é necessário criar um arquivo objeto e um executável para poder rodar um programa. O Python utiliza o próprio código fonte para interpretar e executar os comandos.
- O tempo necessário para efetuar testes é mais rápido.
- A execução do programa é mais lenta comparada com linguagens compiladas, como C.
- Não é conveniente para construir programas que exijam uma estrutura de dados robusta.
- O tempo usado para construir um programa em Python para a maioria de tarefas simples é muito menor do que em outras linguagens compiladas.
- Tem uma vastíssima comunidade, que além de participar tirando dúvidas de outros usuários, constroem bibliotecas vastas, que podem ser instaladas em uma linha de comando.

# O shell do python

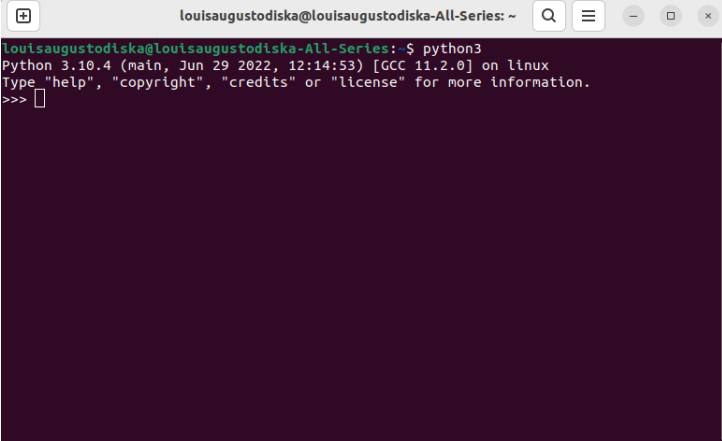
Python possui um shell bastante simples. Abra um terminal (ou konsole - caso do Debian, ou prompt de comando - caso do Windows) e digite `python3`, e veja esta tela:



```
louisaugustodiska@louisaugustodiska-All-Series: ~  
louisaugustodiska@louisaugustodiska-All-Series: $ python3  
Python 3.10.4 (main, Jun 29 2022, 12:14:53) [GCC 11.2.0] on linux  
Type "help", "copyright", "credits" or "license" for more information.  
>>> █
```

# O shell do python

Python possui um shell bastante simples. Abra um terminal (ou konsole - caso do Debian, ou prompt de comando - caso do Windows) e digite python3, e veja esta tela:

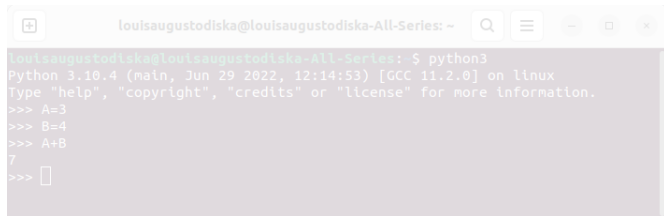


```
louisaugustodiska@louisaugustodiska-All-Series: ~  
louisaugustodiska@louisaugustodiska-All-Series:~$ python3  
Python 3.10.4 (main, Jun 29 2022, 12:14:53) [GCC 11.2.0] on linux  
Type "help", "copyright", "credits" or "license" for more information.  
>>> █
```

# O shell do python

Digite algo como:

```
>>> A = 3
>>> B = 4
>>> A+B
```



```
louisaugustodiska@louisaugustodiska-All-Series: ~
louisaugustodiska@louisaugustodiska-All-Series: $ python3
Python 3.10.4 (main, Jun 29 2022, 12:14:53) [GCC 11.2.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> A=3
>>> B=4
>>> A+B
7
>>> █
```

Em outras palavras, você pode usar o python como uma calculadora rápida se precisar, e tiver acesso a um computador com a linguagem instalada.

Veja mais em [https://youtu.be/FCmCSzjB\\_JU](https://youtu.be/FCmCSzjB_JU)

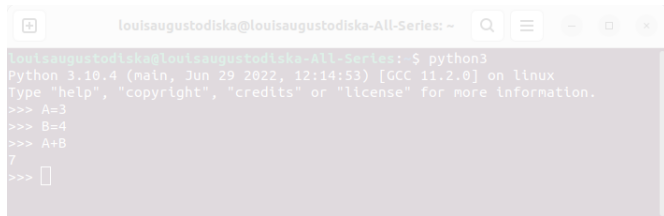
# O shell do python

Digite algo como:

```
>>> A = 3
```

```
>>> B = 4
```

```
>>> A+B
```

A screenshot of a terminal window with a title bar that reads "louisaugustodiska@louisaugustodiska-All-Series: ~". The terminal content shows a user running "python3", which opens a Python 3.10.4 shell. The user enters "A=3", "B=4", and "A+B", with the shell outputting "7".

```
louisaugustodiska@louisaugustodiska-All-Series: ~  
louisaugustodiska@louisaugustodiska-All-Series: $ python3  
Python 3.10.4 (main, Jun 29 2022, 12:14:53) [GCC 11.2.0] on linux  
Type "help", "copyright", "credits" or "license" for more information.  
>>> A=3  
>>> B=4  
>>> A+B  
7  
>>> █
```

Em outras palavras, você pode usar o python como uma calculadora rápida se precisar, e tiver acesso a um computador com a linguagem instalada.

Veja mais em [https://youtu.be/FCmCSzjB\\_JU](https://youtu.be/FCmCSzjB_JU)

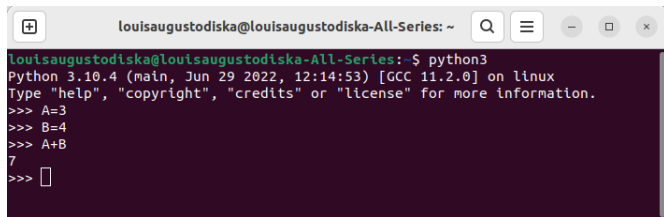
# O shell do python

Digite algo como:

```
>>> A = 3
```

```
>>> B = 4
```

```
>>> A+B
```



```
louisaugustodiska@louisaugustodiska-All-Series: ~  
louisaugustodiska@louisaugustodiska-All-Series:~$ python3  
Python 3.10.4 (main, Jun 29 2022, 12:14:53) [GCC 11.2.0] on linux  
Type "help", "copyright", "credits" or "license" for more information.  
>>> A=3  
>>> B=4  
>>> A+B  
7  
>>> □
```

Em outras palavras, você pode usar o python como uma calculadora rápida se precisar, e tiver acesso a um computador com a linguagem instalada.

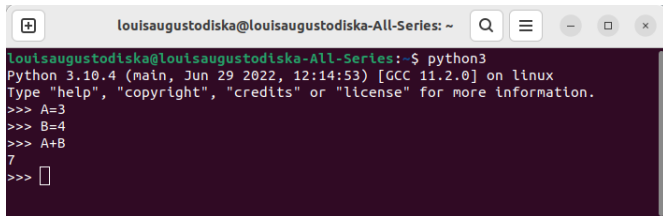
Veja mais em [https://youtu.be/FCmCSzjB\\_JU](https://youtu.be/FCmCSzjB_JU)



# O shell do python

Digite algo como:

```
>>> A = 3
>>> B = 4
>>> A+B
```



```
louisaugustodiska@louisaugustodiska-All-Series: ~
louisaugustodiska@louisaugustodiska-All-Series:~$ python3
Python 3.10.4 (main, Jun 29 2022, 12:14:53) [GCC 11.2.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> A=3
>>> B=4
>>> A+B
7
>>> □
```

Em outras palavras, você pode usar o python como uma calculadora rápida se precisar, e tiver acesso a um computador com a linguagem instalada.

Veja mais em [https://youtu.be/FCmCSzjB\\_JU](https://youtu.be/FCmCSzjB_JU)

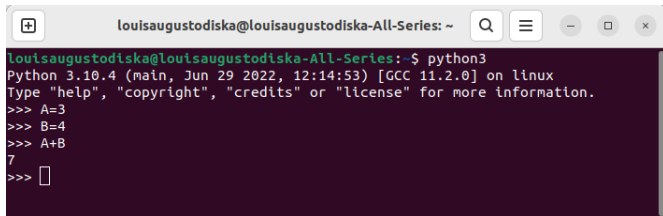
# O shell do python

Digite algo como:

```
>>> A = 3
```

```
>>> B = 4
```

```
>>> A+B
```

A screenshot of a terminal window with a dark background. The window title is "louisaugustodiska@louisaugustodiska-All-Series: ~". The terminal shows the following text:

```
louisaugustodiska@louisaugustodiska-All-Series:~$ python3
Python 3.10.4 (main, Jun 29 2022, 12:14:53) [GCC 11.2.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> A=3
>>> B=4
>>> A+B
7
>>> █
```

Em outras palavras, você pode usar o python como uma calculadora rápida se precisar, e tiver acesso a um computador com a linguagem instalada.

Veja mais em [https://youtu.be/FCmCSzjB\\_JU](https://youtu.be/FCmCSzjB_JU)

# Guardando o código feito

Quando você quer guardar o que foi feito e eventualmente usar em outra situação sem ter de digitar tudo de novo você pode usar um script.

Vamos a um exemplo fácil: resolver uma equação simples do segundo grau com raízes reais. Utilize as variáveis `x1` e `x2` para as soluções.

\*  
\*  
\*

Após o código feito:

\*  
\*  
\*

Insira a linha de código:

```
print("x1 = ", x1, ", x2 = ", x2)
```

para imprimir a solução na tela.

Volte ao terminal e digite `python3 nomedoarquivo.py`.

Pronto!! Você fez seu primeiro programa em python, e não foi um "Olá mundo"

# Guardando o código feito

Quando você quer guardar o que foi feito e eventualmente usar em outra situação sem ter de digitar tudo de novo você pode usar um script.

Vamos a um exemplo fácil: resolver uma equação simples do segundo grau com raízes reais. Utilize as variáveis `x1` e `x2` para as soluções.

\*  
\*  
\*

Após o código feito:

\*  
\*  
\*

Insira a linha de código:

```
print("x1 = ", x1, ", x2 = ", x2)
```

para imprimir a solução na tela.

Volte ao terminal e digite `python3 nomedoarquivo.py`.

Pronto!! Você fez seu primeiro programa em python, e não foi um "Olá mundo"

# Guardando o código feito

Quando você quer guardar o que foi feito e eventualmente usar em outra situação sem ter de digitar tudo de novo você pode usar um script.

Vamos a um exemplo fácil: resolver uma equação simples do segundo grau com raízes reais. Utilize as variáveis `x1` e `x2` para as soluções.

\*  
\*  
\*

Após o código feito:

\*  
\*  
\*

Insira a linha de código:

```
print("x1 = ", x1, ", x2 = ", x2)
```

para imprimir a solução na tela.

Volte ao terminal e digite `python3 nomedoarquivo.py`.

Pronto!! Você fez seu primeiro programa em python, e não foi um "Olá mundo"

# Guardando o código feito

Quando você quer guardar o que foi feito e eventualmente usar em outra situação sem ter de digitar tudo de novo você pode usar um script.

Vamos a um exemplo fácil: resolver uma equação simples do segundo grau com raízes reais. Utilize as variáveis `x1` e `x2` para as soluções.

\*  
\*  
\*

Após o código feito:

\*  
\*  
\*

Insira a linha de código:

```
print("x1 = ", x1, ", x2 = ", x2)
```

para imprimir a solução na tela.

Volte ao terminal e digite `python3 nomedoarquivo.py`.

Pronto!! Você fez seu primeiro programa em python, e não foi um "Olá mundo"

# Guardando o código feito

Quando você quer guardar o que foi feito e eventualmente usar em outra situação sem ter de digitar tudo de novo você pode usar um script.

Vamos a um exemplo fácil: resolver uma equação simples do segundo grau com raízes reais. Utilize as variáveis `x1` e `x2` para as soluções.

\*  
\*  
\*

Após o código feito:

\*  
\*  
\*

Insira a linha de código:

```
print("x1 = ", x1, ", x2 = ", x2)
```

para imprimir a solução na tela.

Volte ao terminal e digite `python3 nomedoarquivo.py`.

Pronto!! Você fez seu primeiro programa em python, e não foi um "Olá mundo"

# Guardando o código feito

Quando você quer guardar o que foi feito e eventualmente usar em outra situação sem ter de digitar tudo de novo você pode usar um script.

Vamos a um exemplo fácil: resolver uma equação simples do segundo grau com raízes reais. Utilize as variáveis `x1` e `x2` para as soluções.

\*  
\*  
\*

Após o código feito:

\*  
\*  
\*

Insira a linha de código:

```
print("x1 = ", x1, ", x2 = ", x2)
```

para imprimir a solução na tela.

Volte ao terminal e digite `python3 nomedoarquivo.py`.

Pronto!! Você fez seu primeiro programa em python, e não foi um "Olá mundo"



# Guardando o código feito

Quando você quer guardar o que foi feito e eventualmente usar em outra situação sem ter de digitar tudo de novo você pode usar um script.

Vamos a um exemplo fácil: resolver uma equação simples do segundo grau com raízes reais. Utilize as variáveis `x1` e `x2` para as soluções.

\*  
\*  
\*

Após o código feito:

\*  
\*  
\*

Insira a linha de código:

```
print("x1 = ", x1, ", x2 = ", x2)
```

para imprimir a solução na tela.

Volte ao terminal e digite `python3 nomedoarquivo.py`.

Pronto!! Você fez seu primeiro programa em python, e não foi um "Olá mundo"

## 1 Python é a linguagem para aprender a programar

- Python é bem simples
- Python é uma linguagem não tipada
- Comentários em Python
- Codificação dos caracteres usados

## 2 Entrada e saída básica de dados

- Entrada de dados
- Saída de dados
- Método format
- Exercícios

## 3 Funções - parte 1

- Guardando códigos para reuso
- Retorno de funções
- Argumentos simples para funções

# Variáveis em Python

Há várias variáveis em Python, às vezes a linguagem automaticamente detecta, às vezes precisamos deixar claro para o python a variável que queremos.

int

float

complex

string

\*  
\*  
\*  
\*  
\*  
\*  
\*  
\*  
\*  
\*  
\*

# Variáveis em Python

Há várias variáveis em Python, às vezes a linguagem automaticamente detecta, às vezes precisamos deixar claro para o python a variável que queremos.

int

float

complex

string

\*  
\*  
\*  
\*  
\*  
\*  
\*  
\*  
\*  
\*  
\*

# Variáveis em Python

Há várias variáveis em Python, às vezes a linguagem automaticamente detecta, às vezes precisamos deixar claro para o python a variável que queremos.

int

float

complex

string

\*  
\*  
\*  
\*  
\*  
\*  
\*  
\*  
\*  
\*  
\*

# Variáveis em Python

Há várias variáveis em Python, às vezes a linguagem automaticamente detecta, às vezes precisamos deixar claro para o python a variável que queremos.

int

float

complex

string

\*  
\*  
\*  
\*  
\*  
\*  
\*  
\*  
\*  
\*  
\*

## 1 Python é a linguagem para aprender a programar

- Python é bem simples
- Python é uma linguagem não tipada
- **Comentários em Python**
- Codificação dos caracteres usados

## 2 Entrada e saída básica de dados

- Entrada de dados
- Saída de dados
- Método format
- Exercícios

## 3 Funções - parte 1

- Guardando códigos para reuso
- Retorno de funções
- Argumentos simples para funções

Comentários são importantes para trazer clareza ao código.

Tudo o que você faz hoje pode não ser muito claro amanhã, então utilize comentários para tudo o que precisar explicar.

Há duas formas de se colocar comentários em python.

**Linha simples** Usamos uma cerquilha #, tudo o que tiver depois da cerquilha é considerado comentário em python.

**Várias linhas** Usamos 3 aspas ' ' ' marcando o início e o final do comentário.

Vamos a alguns exemplos:

```
print("Digite seu nome ") #Pede o nome do usuário.  
Nome = input()  
#Faz a leitura do nome e o guarda na variável Nome.
```



# Comentários em Python

Comentários são importantes para trazer clareza ao código. Tudo o que você faz hoje pode não ser muito claro amanhã, então utilize comentários para tudo o que precisar explicar.

Há duas formas de se colocar comentários em python.

**Linha simples** Usamos uma cerquilha #, tudo o que tiver depois da cerquilha é considerado comentário em python.

**Várias linhas** Usamos 3 aspas ' ' ' marcando o início e o final do comentário.

Vamos a alguns exemplos:

```
print("Digite seu nome ") #Pede o nome do usuário.  
Nome = input()  
#Faz a leitura do nome e o guarda na variável Nome.
```

# Comentários em Python

Comentários são importantes para trazer clareza ao código. Tudo o que você faz hoje pode não ser muito claro amanhã, então utilize comentários para tudo o que precisar explicar.

Há duas formas de se colocar comentários em python.

**Linha simples** Usamos uma cerquilha #, tudo o que tiver depois da cerquilha é considerado comentário em python.

**Várias linhas** Usamos 3 aspas ''' marcando o início e o final do comentário.

Vamos a alguns exemplos:

```
print("Digite seu nome ") #Pede o nome do usuário.  
Nome = input()  
#Faz a leitura do nome e o guarda na variável Nome.
```

# Comentários em Python

Comentários são importantes para trazer clareza ao código. Tudo o que você faz hoje pode não ser muito claro amanhã, então utilize comentários para tudo o que precisar explicar.

Há duas formas de se colocar comentários em python.

**Linha simples** Usamos uma cerquilha #, tudo o que tiver depois da cerquilha é considerado comentário em python.

**Várias linhas** Usamos 3 aspas ''' marcando o início e o final do comentário.

Vamos a alguns exemplos:

```
print("Digite seu nome ") #Pede o nome do usuário.  
Nome = input()  
#Faz a leitura do nome e o guarda na variável Nome.
```

# Comentários em Python

Comentários são importantes para trazer clareza ao código. Tudo o que você faz hoje pode não ser muito claro amanhã, então utilize comentários para tudo o que precisar explicar.

Há duas formas de se colocar comentários em python.

**Linha simples** Usamos uma cerquilha #, tudo o que tiver depois da cerquilha é considerado comentário em python.

**Várias linhas** Usamos 3 aspas ''' marcando o início e o final do comentário.

Vamos a alguns exemplos:

```
print("Digite seu nome ") #Pede o nome do usuário.  
Nome = input()  
#Faz a leitura do nome e o guarda na variável Nome.
```

# Comentários em Python

Comentários são importantes para trazer clareza ao código. Tudo o que você faz hoje pode não ser muito claro amanhã, então utilize comentários para tudo o que precisar explicar.

Há duas formas de se colocar comentários em python.

**Linha simples** Usamos uma cerquilha #, tudo o que tiver depois da cerquilha é considerado comentário em python.

**Várias linhas** Usamos 3 aspas ''' marcando o início e o final do comentário.

Vamos a alguns exemplos:

```
print("Digite seu nome ") #Pede o nome do usuário.  
Nome = input()  
#Faz a leitura do nome e o guarda na variável Nome.
```

## 1 Python é a linguagem para aprender a programar

- Python é bem simples
- Python é uma linguagem não tipada
- Comentários em Python
- **Codificação dos caracteres usados**

## 2 Entrada e saída básica de dados

- Entrada de dados
- Saída de dados
- Método format
- Exercícios

## 3 Funções - parte 1

- Guardando códigos para reuso
- Retorno de funções
- Argumentos simples para funções

## Linha shebang

É uma linha colocada no início do programa em Python que serve para indicar a acentuação usada. Para línguas latinas, com til, cedilha, crase e acentos:

```
#!/usr/bin/env python
```

Apesar de iniciar com cerquilha, não é um comentário, e sim uma informação à linguagem de que se está usando `utf-8`, que é uma codificação de caracteres que contém os caracteres latinos.

Cirílico (caracteres eslavos) tem codificação `ISO 8859-5`, então russos, ucranianos, moldavos etc. utilizam linha shebang apropriada aos caracteres eslavos.

## Linha shebang

É uma linha colocada no início do programa em Python que serve para indicar a acentuação usada. Para línguas latinas, com til, cedilha, crase e acentos:

```
#!/usr/bin/env python
```

Apesar de iniciar com cerquilha, não é um comentário, e sim uma informação à linguagem de que se está usando `utf-8`, que é uma codificação de caracteres que contém os caracteres latinos.

Cirílico (caracteres eslavos) tem codificação `ISO 8859-5`, então russos, ucranianos, moldavos etc. utilizam linha shebang apropriada aos caracteres eslavos.



## Linha shebang

É uma linha colocada no início do programa em Python que serve para indicar a acentuação usada. Para línguas latinas, com til, cedilha, crase e acentos:

```
#-*- coding: utf-8 -*-
```

Apesar de iniciar com cerquilha, não é um comentário, e sim uma informação à linguagem de que se está usando `utf-8`, que é uma codificação de caracteres que contém os caracteres latinos.

Cirílico (caracteres eslavos) tem codificação `ISO 8859-5`, então russos, ucranianos, moldavos etc. utilizam linha shebang apropriada aos caracteres eslavos.

## 1 Python é a linguagem para aprender a programar

- Python é bem simples
- Python é uma linguagem não tipada
- Comentários em Python
- Codificação dos caracteres usados

## 2 Entrada e saída básica de dados

- **Entrada de dados**
- Saída de dados
- Método format
- Exercícios

## 3 Funções - parte 1

- Guardando códigos para reuso
- Retorno de funções
- Argumentos simples para funções

# Entrada de dados

A função `input()` faz uma pausa no programa e espera uma entrada do usuário pelo terminal.

`input()` lê essa entrada como uma string, portanto, se a entrada esperada for um número ela deve ser convertida usando-se as funções de conversão `int()` ou `float()`.

```
#!/usr/bin/env python3
#-*- coding: utf-8 -*-
print('Programa para calcular a media de um aluno:')
nome = input('Entre com o nome do aluno: ')
nota1 = float(input("Entre com a primeira nota: "))
nota2 = float(input("Entre com a segunda nota: "))
media = (nota1 + nota2)/2
print(nome, 'teve media igual a:', media)
```

Que possui a execução dada por:

```
Programa para calcular a media de um aluno:
Entre com o nome do aluno: Jacinto
Entre com a primeira nota: 4
Entre com a segunda nota: 6
Jacinto teve media igual a: 5.0
```

# Entrada de dados

A função `input()` faz uma pausa no programa e espera uma entrada do usuário pelo terminal.

`input()` lê essa entrada como uma string, portanto, se a entrada esperada for um número ela deve ser convertida usando-se as funções de conversão `int()` ou `float()`.

```
#!/usr/bin/env python3
#-*- coding: utf-8 -*-
print('Programa para calcular a media de um aluno:')
nome = input('Entre com o nome do aluno: ')
nota1 = float(input("Entre com a primeira nota: "))
nota2 = float(input("Entre com a segunda nota: "))
media = (nota1 + nota2)/2
print(nome, 'teve media igual a:', media)
```

Que possui a execução dada por:

```
Programa para calcular a media de um aluno:
Entre com o nome do aluno: Jacinto
Entre com a primeira nota: 4
Entre com a segunda nota: 6
Jacinto teve media igual a: 5.0
```

# Entrada de dados

A função `input()` faz uma pausa no programa e espera uma entrada do usuário pelo terminal.

`input()` lê essa entrada como uma string, portanto, se a entrada esperada for um número ela deve ser convertida usando-se as funções de conversão `int()` ou `float()`.

```
#!/usr/bin/env python3
#-*- coding: utf-8 -*-
print('Programa para calcular a media de um aluno:')
nome = input('Entre com o nome do aluno: ')
nota1 = float(input("Entre com a primeira nota: "))
nota2 = float(input("Entre com a segunda nota: "))
media = (nota1 + nota2)/2
print(nome, 'teve media igual a:', media)
```

Que possui a execução dada por:

```
Programa para calcular a media de um aluno:
Entre com o nome do aluno: Jacinto
Entre com a primeira nota: 4
Entre com a segunda nota: 6
Jacinto teve media igual a: 5.0
```

## 1 Python é a linguagem para aprender a programar

- Python é bem simples
- Python é uma linguagem não tipada
- Comentários em Python
- Codificação dos caracteres usados

## 2 Entrada e saída básica de dados

- Entrada de dados
- **Saída de dados**
- Método format
- Exercícios

## 3 Funções - parte 1

- Guardando códigos para reuso
- Retorno de funções
- Argumentos simples para funções

# Saída de dados

A função `print()` serve para imprimir os argumentos passados a ela no terminal. Sua forma mais simples é:

```
CelsiusFahr.py ✖
#-*- coding: utf-8 -*-
# Programa para converter graus de Celsius para Fahrenheit

c = 25
f = 1.8*c + 32
print(c, ' graus Celsius = ', f, ' graus Fahrenheit')
```

Que possui a execução dada por:



```
louisaugustodiska@louisaugustodiska-All-Series: /media/louisaugust...
louisaugustodiska@louisaugustodiska-All-Series: /media/louisaugustodiska/2e4ada13-e40e-4
082-9348-b615e67393fd/home/louisaugustodiska/MEGA/MEGAsync/IFSC/Python/Aulas_slides
/Encontro_1 - Apresentacao_da_Linguagem_(vazio_alada)/codigo$ python3 CelsiusFahr.py
25 graus Celsius = 77.0 graus Fahrenheit
```

# Saída de dados

A função `print()` serve para imprimir os argumentos passados a ela no terminal. Sua forma mais simples é:

```
CelsiusFahr.py ✖
#-*- coding: utf-8 -*-
# Programa para converter graus de Celsius para Fahrenheit

c = 25
f = 1.8*c + 32
print(c, ' graus Celsius = ', f, ' graus Fahrenheit')
```

Que possui a execução dada por:

```
louisaugustodiska@louisaugustodiska-All-Series: /media/louisaugust...
louisaugustodiska@louisaugustodiska-All-Series: /media/louisaugustodiska/2e4ada13-e40e-4
082-9348-b615e67393fd/home/louisaugustodiska/MEGA/MEGAsync/IFSC/Python/Aulas_slides
/Encontro_1 - Apresentacao da linguagem (bazilo_alada)/codigo$ python3 CelsiusFahr.py
25 graus Celsius = 77.0 graus Fahrenheit
```



# Saída de dados

A função `print()` serve para imprimir os argumentos passados a ela no terminal. Sua forma mais simples é:

```
CelsiusFahr.py ✖
#-*- coding: utf-8 -*-
# Programa para converter graus de Celsius para Fahrenheit

c = 25
f = 1.8*c + 32
print(c, ' graus Celsius = ', f, ' graus Fahrenheit')
```

Que possui a execução dada por:

```
louisaugustodiska@louisaugustodiska-All-Series: /media/louisaugust...
louisaugustodiska@louisaugustodiska-All-Series:~/media/louisaugustodiska/2e4ada13-e40e-4062-9348-b615e67393fd/home/louisaugustomaindiskb/MEGA/MEGAsync/IFSC/Python/Aulas Slides/Encontro 1 - Apresentacao da linguagem (vazio ainda)/codigos$ python3 CelsiusFahr.py
25 graus Celsius = 77.0 graus Fahrenheit
```

# Saída de dados

A função `print` pode receber vários argumentos separados por vírgulas e imprimi-los no terminal.

Dentre os argumentos que `print` pode receber existem três parâmetros que são passados ao final com as palavras-chave `sep`, `end` e `file`.

Caso se deixe em branco estes argumentos, eles utilizam argumentos padrão.

<code>sep</code>	<i>valor padrão: espaço em branco.</i> Quando dois ou mais argumentos são passados para a função <code>print</code> , <code>sep</code> coloca entre eles um espaço em branco ou um marcador escolhido.
<code>end</code>	<i>valor padrão: linha em branco.</i> Após a saída automaticamente se adiciona uma linha, mas pode-se trocar isto por nada, "", ou uma tabulação, <code>\t</code> , ou mesmo um texto (entre aspas).
<code>file</code>	<i>valor padrão: terminal.</i> Pode-se salvar em um arquivo também, mas é necessário abrir o arquivo separadamente. Será estudado à parte.



A função `print` pode receber vários argumentos separados por vírgulas e imprimi-los no terminal.

Dentre os argumentos que `print` pode receber existem três parâmetros que são passados ao final com as palavras-chave `sep`, `end` e `file`.

Caso se deixe em branco estes argumentos, eles utilizam argumentos padrão.

<code>sep</code>	<i>valor padrão: espaço em branco.</i> Quando dois ou mais argumentos são passados para a função <code>print</code> , <code>sep</code> coloca entre eles um espaço em branco ou um marcador escolhido.
<code>end</code>	<i>valor padrão: linha em branco.</i> Após a saída automaticamente se adiciona uma linha, mas pode-se trocar isto por nada, "", ou uma tabulação, <code>\t</code> , ou mesmo um texto (entre aspas).
<code>file</code>	<i>valor padrão: terminal.</i> Pode-se salvar em um arquivo também, mas é necessário abrir o arquivo separadamente. Será estudado à parte.

# Saída de dados

Observe o programa abaixo e sua saída:

```
#!/*- coding: utf-8 -*-
ano1 = '1980'
ano2 = '1990'
ano3 = '2000'
ano4 = '2010'
texto = "Alterando o valor de sep"
print(texto)
print(ano1, ano2, ano3, ano4, sep='--->')
texto = "Alterando o valor de sep e end"
print(texto)
print(ano1, ano2, ano3, ano4, sep='--->', end='...\n')
print("Programa ", __name__, " terminado")
```

```
Alterando o valor de sep
1980--->1990--->2000--->2010
Alterando o valor de sep e end
1980--->1990--->2000--->2010...
Programa __main__ terminado
```

# Saída de dados

Observe o programa abaixo e sua saída:

```
#-*- coding: utf-8 -*-  
ano1 = '1980'  
ano2 = '1990'  
ano3 = '2000'  
ano4 = '2010'  
texto = "Alterando o valor de sep"  
print(texto)  
print(ano1, ano2, ano3, ano4, sep='--->')  
texto = "Alterando o valor de sep e end"  
print(texto)  
print(ano1, ano2, ano3, ano4, sep='--->', end='...\n')  
print("Programa ", __name__, " terminado")
```

```
Alterando o valor de sep  
1980--->1990--->2000--->2010  
Alterando o valor de sep e end  
1980--->1990--->2000--->2010...  
Programa __main__ terminado
```

## 1 Python é a linguagem para aprender a programar

- Python é bem simples
- Python é uma linguagem não tipada
- Comentários em Python
- Codificação dos caracteres usados

## 2 Entrada e saída básica de dados

- Entrada de dados
- Saída de dados
- **Método format**
- Exercícios

## 3 Funções - parte 1

- Guardando códigos para reuso
- Retorno de funções
- Argumentos simples para funções

# Método format

O método `format()` serve para criar uma string que contém campos entre chaves a serem substituídos pelos argumentos de `format`.

```
#!/*- coding: utf-8 -*-
str = 'O filme {0} merece {1} oscars'
str2 =str.format('Monty Python', 2)
print(str2)
#Podemos usar para especificar número de vírgulas
import numpy as np
print("Pi com muitas casas decimais: ", np.pi)
print("Pi com {0} casas decimais: {1:.3f}".format(3,np.pi))
```

A string `str2` vai receber a string `str`, tendo trocado o valor de `{0}` por `Monty Python` e de `{1}` por `2`.

Usando `{1:.3f}` fazemos com que a segunda especificação seja de um número float até a terceira casa depois da vírgula.

```
O filme Monty Python merece 2 oscars
Pi com muitas casas decimais: 3.141592653589793
Pi com 3 casas decimais: 3.142
```



# Método format

O método `format()` serve para criar uma string que contem campos entre chaves a serem substituídos pelos argumentos de `format`.

```
#-*- coding: utf-8 -*-
str = 'O filme {0} merece {1} oscars'
str2 =str.format('Monty Python', 2)
print(str2)
#Podemos usar para especificar número de vírgulas
import numpy as np
print("Pi com muitas casas decimais: ", np.pi)
print("Pi com {0} casas decimais: {1:.3f}".format(3,np.pi))
```

A string `str2` vai receber a string `str`, tendo trocado o valor de `{0}` por `Monty Python` e de `{1}` por `2`.

Usando `{1:.3f}` fazemos com que a segunda especificação seja de um número float até a terceira casa depois da vírgula.

```
O filme Monty Python merece 2 oscars
Pi com muitas casas decimais: 3.141592653589793
Pi com 3 casas decimais: 3.142
```

# Método format

O método `format()` serve para criar uma string que contém campos entre chaves a serem substituídos pelos argumentos de `format`.

```
#-*- coding: utf-8 -*-  
str = 'O filme {0} merece {1} oscars'  
str2 =str.format('Monty Python', 2)  
print(str2)  
#Podemos usar para especificar número de vírgulas  
import numpy as np  
print("Pi com muitas casas decimais: ", np.pi)  
print("Pi com {0} casas decimais: {1:.3f}".format(3,np.pi))
```

A string `str2` vai receber a string `str`, tendo trocado o valor de `{0}` por `Monty Python` e de `{1}` por `2`.

Usando `{1:.3f}` fazemos com que a segunda especificação seja de um número float até a terceira casa depois da vírgula.

```
O filme Monty Python merece 2 oscars  
Pi com muitas casas decimais: 3.141592653589793  
Pi com 3 casas decimais: 3.142
```

# Método format

O método `format()` serve para criar uma string que contém campos entre chaves a serem substituídos pelos argumentos de `format`.

```
#-*- coding: utf-8 -*-
str = 'O filme {0} merece {1} oscars'
str2 =str.format('Monty Python', 2)
print(str2)
#Podemos usar para especificar número de vírgulas
import numpy as np
print("Pi com muitas casas decimais: ", np.pi)
print("Pi com {0} casas decimais: {1:.3f}".format(3,np.pi))
```

A string `str2` vai receber a string `str`, tendo trocado o valor de `{0}` por `Monty Python` e de `{1}` por `2`.

Usando `{1:.3f}` fazemos com que a segunda especificação seja de um número float até a terceira casa depois da vírgula.

```
O filme Monty Python merece 2 oscars
Pi com muitas casas decimais: 3.141592653589793
Pi com 3 casas decimais: 3.142
```

## 1 Python é a linguagem para aprender a programar

- Python é bem simples
- Python é uma linguagem não tipada
- Comentários em Python
- Codificação dos caracteres usados

## 2 Entrada e saída básica de dados

- Entrada de dados
- Saída de dados
- Método format
- Exercícios

## 3 Funções - parte 1

- Guardando códigos para reuso
- Retorno de funções
- Argumentos simples para funções

# Equação do segundo grau

Vamos resolver uma simples equação do segundo grau em python:

$$x^2 - 5x + 4 = 0:$$

```
#-*- coding: utf-8 -*-  
a=1  
b = -5  
c = 4  
Delta = b**2-4*a*c  
x1 = (-b-Delta**0.5) / (2*a)  
x2 = (-b+Delta**0.5) / (2*a)  
print("Soluções: x1 = ", x1, ", x2 = ", x2)
```

Observe que no código os valores dos coeficientes são fixos, e que o script somente resolve esta equação, específica.

Todavia temos algo interessante, não é preciso dizer qual o tipo das variáveis.

Refaça o código para a equação  $x^2 + 2 = 0$ .

# Equação do segundo grau

Vamos resolver uma simples equação do segundo grau em python:

$$x^2 - 5x + 4 = 0:$$

```
#-*- coding: utf-8 -*-  
a=1  
b = -5  
c = 4  
Delta = b**2-4*a*c  
x1 = (-b-Delta**0.5) / (2*a)  
x2 = (-b+Delta**0.5) / (2*a)  
print("Soluções: x1 = ", x1, ", x2 = ", x2)
```

Observe que no código os valores dos coeficientes são fixos, e que o script somente resolve esta equação, específica.

Todavia temos algo interessante, não é preciso dizer qual o tipo das variáveis.

Refaça o código para a equação  $x^2 + 2 = 0$ .

# Equação do segundo grau

Vamos resolver uma simples equação do segundo grau em python:

$$x^2 - 5x + 4 = 0:$$

```
#-*- coding: utf-8 -*-  
a=1  
b = -5  
c = 4  
Delta = b**2-4*a*c  
x1 = (-b-Delta**0.5) / (2*a)  
x2 = (-b+Delta**0.5) / (2*a)  
print("Soluções: x1 = ", x1, ", x2 = ", x2)
```

Observe que no código os valores dos coeficientes são fixos, e que o script somente resolve esta equação, específica.

Todavia temos algo interessante, não é preciso dizer qual o tipo das variáveis.

Refaça o código para a equação  $x^2 + 2 = 0$ .

# Equação do segundo grau

Vamos resolver uma simples equação do segundo grau em python:

$$x^2 - 5x + 4 = 0:$$

```
#-*- coding: utf-8 -*-  
a=1  
b = -5  
c = 4  
Delta = b**2-4*a*c  
x1 = (-b-Delta**0.5) / (2*a)  
x2 = (-b+Delta**0.5) / (2*a)  
print("Soluções: x1 = ", x1, ", x2 = ", x2)
```

Observe que no código os valores dos coeficientes são fixos, e que o script somente resolve esta equação, específica.

Todavia temos algo interessante, não é preciso dizer qual o tipo das variáveis.

Refaça o código para a equação  $x^2 + 2 = 0$ .



# Equação do segundo grau

Vamos resolver uma simples equação do segundo grau em python:

$$x^2 - 5x + 4 = 0:$$

```
#-*- coding: utf-8 -*-  
a=1  
b = -5  
c = 4  
Delta = b**2-4*a*c  
x1 = (-b-Delta**0.5) / (2*a)  
x2 = (-b+Delta**0.5) / (2*a)  
print("Soluções: x1 = ", x1, ", x2 = ", x2)
```

Observe que no código os valores dos coeficientes são fixos, e que o script somente resolve esta equação, específica.

Todavia temos algo interessante, não é preciso dizer qual o tipo das variáveis.

Refaça o código para a equação  $x^2 + 2 = 0$ .

# Equação do segundo grau

Temos as saídas:

1º script:

Soluções:  $x_1 = 1.0$  ,  $x_2 = 4.0$

2º script:

Soluções:  $x_1 = -1.4142135623730951$  ,  $x_2 = 1.4142135623730951$

Observe que o python não dá as funções com resultados simbólicos,  $x_1 = \sqrt{2}$  e  $x_2 = -\sqrt{2}$ , mas sim resultados numéricos.

Como a linguagem é não tipada, o python tem que decidir qual o tipo da variável, no caso a resposta é dada em float nos dois casos porque houve o cálculo de uma raiz quadrada.

Refaça o script colocando saidas com três casas decimais somente, usando o método `format`.

Vamos agora reescrever o código, mas inserindo os coeficientes via teclado.

# Equação do segundo grau

Temos as saídas:

1º script:

Soluções:  $x_1 = 1.0$  ,  $x_2 = 4.0$

2º script:

Soluções:  $x_1 = -1.4142135623730951$  ,  $x_2 = 1.4142135623730951$

Observe que o python não dá as funções com resultados simbólicos,  $x_1 = \sqrt{2}$  e  $x_2 = -\sqrt{2}$ , mas sim resultados numéricos.

Como a linguagem é não tipada, o python tem que decidir qual o tipo da variável, no caso a resposta é dada em float nos dois casos porque houve o cálculo de uma raiz quadrada.

Refaça o script colocando saidas com três casas decimais somente, usando o método `format`.

Vamos agora reescrever o código, mas inserindo os coeficientes via teclado.

# Equação do segundo grau

Temos as saídas:

1º script:

Soluções:  $x_1 = 1.0$  ,  $x_2 = 4.0$

2º script:

Soluções:  $x_1 = -1.4142135623730951$  ,  $x_2 = 1.4142135623730951$

Observe que o python não dá as funções com resultados simbólicos,  $x_1 = \sqrt{2}$  e  $x_2 = -\sqrt{2}$ , mas sim resultados numéricos.

Como a linguagem é não tipada, o python tem que decidir qual o tipo da variável, no caso a resposta é dada em float nos dois casos porque houve o cálculo de uma raiz quadrada.

Refaça o script colocando saidas com três casas decimais somente, usando o método `format`.

Vamos agora reescrever o código, mas inserindo os coeficientes via teclado.

# Equação do segundo grau

Temos as saídas:

1º script:

Soluções:  $x_1 = 1.0$  ,  $x_2 = 4.0$

2º script:

Soluções:  $x_1 = -1.4142135623730951$  ,  $x_2 = 1.4142135623730951$

Observe que o python não dá as funções com resultados simbólicos,  $x_1 = \sqrt{2}$  e  $x_2 = -\sqrt{2}$ , mas sim resultados numéricos.

Como a linguagem é não tipada, o python tem que decidir qual o tipo da variável, no caso a resposta é dada em float nos dois casos porque houve o cálculo de uma raiz quadrada.

Refaça o script colocando saídas com três casas decimais somente, usando o método `format`.

Vamos agora reescrever o código, mas inserindo os coeficientes via teclado.

# Equação do segundo grau

Temos as saídas:

1º script:

Soluções:  $x_1 = 1.0$  ,  $x_2 = 4.0$

2º script:

Soluções:  $x_1 = -1.4142135623730951$  ,  $x_2 = 1.4142135623730951$

Observe que o python não dá as funções com resultados simbólicos,  $x_1 = \sqrt{2}$  e  $x_2 = -\sqrt{2}$ , mas sim resultados numéricos.

Como a linguagem é não tipada, o python tem que decidir qual o tipo da variável, no caso a resposta é dada em float nos dois casos porque houve o cálculo de uma raiz quadrada.

Refaça o script colocando saídas com três casas decimais somente, usando o método `format`.

Vamos agora reescrever o código, mas inserindo os coeficientes via teclado.

# Equação do segundo grau

Temos as saídas:

1º script:

Soluções:  $x_1 = 1.0$  ,  $x_2 = 4.0$

2º script:

Soluções:  $x_1 = -1.4142135623730951$  ,  $x_2 = 1.4142135623730951$

Observe que o python não dá as funções com resultados simbólicos,  $x_1 = \sqrt{2}$  e  $x_2 = -\sqrt{2}$ , mas sim resultados numéricos.

Como a linguagem é não tipada, o python tem que decidir qual o tipo da variável, no caso a resposta é dada em float nos dois casos porque houve o cálculo de uma raiz quadrada.

Refaça o script colocando saídas com três casas decimais somente, usando o método `format`.

Vamos agora reescrever o código, mas inserindo os coeficientes via teclado.

# Equação do segundo grau

```
a = input("Entre com o coef a: ")
a = float(a)
b = input("Entre com o coef b: ")
b = float(b)
c = float(input("Entre com o coef c: "))
Delta = b**2-4*a*c
x1 = (-b-Delta**0.5)/(2*a)
x2 = (-b+Delta**0.5)/(2*a)
print("Coeficientes: ", a, b, c)
print("Delta = ", Delta)
print("Soluções: x1 = ", x1, "x2 = ", x2)
```

Observe que agora há mais coisas para fazer. Lembre-se de que a entrada de dados é sempre no tipo string, logo devemos proceder à conversão do tipo para `int` ou `float`.

Os coeficientes  $a$  e  $b$  foram convertidos em dois passos, enquanto o coeficiente  $c$  num passo somente. Teste o programa com vários coeficientes diferentes, e verifique como são as saídas, especialmente para números complexos, exemplo  $a = 1$ ,  $b = 2$ ,  $c = 2$ .



# Equação do segundo grau

```
a = input("Entre com o coef a: ")
a = float(a)
b = input("Entre com o coef b: ")
b = float(b)
c = float(input("Entre com o coef c: "))
Delta = b**2-4*a*c
x1 = (-b-Delta**0.5)/(2*a)
x2 = (-b+Delta**0.5)/(2*a)
print("Coeficientes: ", a, b, c)
print("Delta = ", Delta)
print("Soluções: x1 = ", x1, "x2 = ", x2)
```

Observe que agora há mais coisas para fazer. Lembre-se de que a entrada de dados é sempre no tipo string, logo devemos proceder à conversão do tipo para `int` ou `float`.

Os coeficientes  $a$  e  $b$  foram convertidos em dois passos, enquanto o coeficiente  $c$  num passo somente. Teste o programa com vários coeficientes diferentes, e verifique como são as saídas, especialmente para números complexos, exemplo  $a = 1$ ,  $b = 2$ ,  $c = 2$ .

# Equação do segundo grau

```
a = input("Entre com o coef a: ")
a = float(a)
b = input("Entre com o coef b: ")
b = float(b)
c = float(input("Entre com o coef c: "))
Delta = b**2-4*a*c
x1 = (-b-Delta**0.5)/(2*a)
x2 = (-b+Delta**0.5)/(2*a)
print("Coeficientes: ", a, b, c)
print("Delta = ", Delta)
print("Soluções: x1 = ", x1, "x2 = ", x2)
```

Observe que agora há mais coisas para fazer. Lembre-se de que a entrada de dados é sempre no tipo string, logo devemos proceder à conversão do tipo para `int` ou `float`.

Os coeficientes  $a$  e  $b$  foram convertidos em dois passos, enquanto o coeficiente  $c$  num passo somente. Teste o programa com vários coeficientes diferentes, e verifique como são as saídas, especialmente para números complexos, exemplo  $a = 1$ ,  $b = 2$ ,  $c = 2$ .

## 1 Python é a linguagem para aprender a programar

- Python é bem simples
- Python é uma linguagem não tipada
- Comentários em Python
- Codificação dos caracteres usados

## 2 Entrada e saída básica de dados

- Entrada de dados
- Saída de dados
- Método format
- Exercícios

## 3 Funções - parte 1

- Guardando códigos para reuso
- Retorno de funções
- Argumentos simples para funções

# Guardando códigos para reuso

Tínhamos os seguintes problemas no código anterior:

- A equação do segundo grau só podia ser executada uma vez.
- Ou resolvíamos uma equação fixa, ou precisávamos inserir os dados via teclado.
- O programa não podia selecionar dados para calcular a equação do segundo grau.

Para resolver estes problemas existem as funções.

Nas funções colocamos o código que queremos reutilizar num bloco de código, sendo chamada pelo comando `def` mais um nome, seguido por `()` : . O bloco de código deve estar indentado.

No próximo slide veremos a função para resolução de equação de segundo grau mais simples, somente encapsulando o código usado. A chamada à função é feita na linha `eq2grau()` .

# Guardando códigos para reuso

Tínhamos os seguintes problemas no código anterior:

- A equação do segundo grau só podia ser executada uma vez.
- Ou resolvíamos uma equação fixa, ou precisávamos inserir os dados via teclado.
- O programa não podia selecionar dados para calcular a equação do segundo grau.

Para resolver estes problemas existem as funções.

Nas funções colocamos o código que queremos reutilizar num bloco de código, sendo chamada pelo comando `def` mais um nome, seguido por `()` : . O bloco de código deve estar indentado.

No próximo slide veremos a função para resolução de equação de segundo grau mais simples, somente encapsulando o código usado. A chamada à função é feita na linha `eq2grau()` .

# Guardando códigos para reuso

Tínhamos os seguintes problemas no código anterior:

- A equação do segundo grau só podia ser executada uma vez.
- Ou resolvíamos uma equação fixa, ou precisávamos inserir os dados via teclado.
- O programa não podia selecionar dados para calcular a equação do segundo grau.

Para resolver estes problemas existem as funções.

Nas funções colocamos o código que queremos reutilizar num bloco de código, sendo chamada pelo comando `def` mais um nome, seguido por `()` : . O bloco de código deve estar indentado.

No próximo slide veremos a função para resolução de equação de segundo grau mais simples, somente encapsulando o código usado. A chamada à função é feita na linha `eq2grau()` .

# Guardando códigos para reuso

Tínhamos os seguintes problemas no código anterior:

- A equação do segundo grau só podia ser executada uma vez.
- Ou resolvíamos uma equação fixa, ou precisávamos inserir os dados via teclado.
- O programa não podia selecionar dados para calcular a equação do segundo grau.

Para resolver estes problemas existem as funções.

Nas funções colocamos o código que queremos reutilizar num bloco de código, sendo chamada pelo comando `def` mais um nome, seguido por `()` : . O bloco de código deve estar indentado.

No próximo slide veremos a função para resolução de equação de segundo grau mais simples, somente encapsulando o código usado. A chamada à função é feita na linha `eq2grau()` .

# Guardando códigos para reuso

Tínhamos os seguintes problemas no código anterior:

- A equação do segundo grau só podia ser executada uma vez.
- Ou resolvíamos uma equação fixa, ou precisávamos inserir os dados via teclado.
- O programa não podia selecionar dados para calcular a equação do segundo grau.

Para resolver estes problemas existem as funções.

Nas funções colocamos o código que queremos reutilizar num bloco de código, sendo chamada pelo comando `def` mais um nome, seguido por `()` : . O bloco de código deve estar indentado.

No próximo slide veremos a função para resolução de equação de segundo grau mais simples, somente encapsulando o código usado. A chamada à função é feita na linha `eq2grau()` .



# Guardando códigos para reuso

Tínhamos os seguintes problemas no código anterior:

- A equação do segundo grau só podia ser executada uma vez.
- Ou resolvíamos uma equação fixa, ou precisávamos inserir os dados via teclado.
- O programa não podia selecionar dados para calcular a equação do segundo grau.

Para resolver estes problemas existem as funções.

Nas funções colocamos o código que queremos reutilizar num bloco de código, sendo chamada pelo comando `def` mais um nome, seguido por `() :`. O bloco de código deve estar indentado.

No próximo slide veremos a função para resolução de equação de segundo grau mais simples, somente encapsulando o código usado. A chamada à função é feita na linha `eq2grau()`.

# Guardando códigos para reuso

Tínhamos os seguintes problemas no código anterior:

- A equação do segundo grau só podia ser executada uma vez.
- Ou resolvíamos uma equação fixa, ou precisávamos inserir os dados via teclado.
- O programa não podia selecionar dados para calcular a equação do segundo grau.

Para resolver estes problemas existem as funções.

Nas funções colocamos o código que queremos reutilizar num bloco de código, sendo chamada pelo comando `def` mais um nome, seguido por `()` : . O bloco de código deve estar indentado.

No próximo slide veremos a função para resolução de equação de segundo grau mais simples, somente encapsulando o código usado. A chamada à função é feita na linha `eq2grau()` .

# Guardando códigos para reuso

```
#-*- coding: utf-8 -*-
def eq2grau():
    #Coeficientes
    a = input("Entre com o coef a: ")
    a = float(a)
    b = input("Entre com o coef b: ")
    b = float(b)
    c = float(input("Entre com o coef c: "))
    Delta = b**2-4*a*c
    x1 = (-b-Delta**0.5)/(2*a)
    x2 = (-b+Delta**0.5)/(2*a)
    print("Coeficientes: ", a, b, c)
    print("Delta = ", Delta)
    print("Soluções: x1 = ", x1, "x2 = ", x2)

eq2grau()
```

## 1 Python é a linguagem para aprender a programar

- Python é bem simples
- Python é uma linguagem não tipada
- Comentários em Python
- Codificação dos caracteres usados

## 2 Entrada e saída básica de dados

- Entrada de dados
- Saída de dados
- Método format
- Exercícios

## 3 Funções - parte 1

- Guardando códigos para reuso
- **Retorno de funções**
- Argumentos simples para funções

# Retorno de funções

As funções podem, além de processar um bloco de código, retornar algum valor.

Suponha que, por alguma razão, quer-se obter valor do discriminante delta da função. Neste exemplo queremos que a função imprima os valores das raízes e retorne o valor de delta para o código que chamou a função.

```
def eq2grau():
    a = float(input("Entre com o coef a: "))
    b = float(input("Entre com o coef b: "))
    c = float(input("Entre com o coef c: "))
    Delta = b**2-4*a*c
    x1 = (-b-Delta**0.5)/(2*a)
    x2 = (-b+Delta**0.5)/(2*a)
    print("Soluções: x1 = ", x1, "x2 = ", x2)
    return Delta

var = eq2grau()
print("O valor de delta é ", var)
```

# Retorno de funções

As funções podem, além de processar um bloco de código, retornar algum valor.

Suponha que, por alguma razão, quer-se obter valor do discriminante delta da função. Neste exemplo queremos que a função imprima os valores das raízes e retorne o valor de delta para o código que chamou a função.

```
def eq2grau():
    a = float(input("Entre com o coef a: "))
    b = float(input("Entre com o coef b: "))
    c = float(input("Entre com o coef c: "))
    Delta = b**2-4*a*c
    x1 = (-b-Delta**0.5)/(2*a)
    x2 = (-b+Delta**0.5)/(2*a)
    print("Soluções: x1 = ", x1, "x2 = ", x2)
    return Delta

var = eq2grau()
print("O valor de delta é ", var)
```

# Retorno de funções

As funções podem, além de processar um bloco de código, retornar algum valor.

Suponha que, por alguma razão, quer-se obter valor do discriminante delta da função. Neste exemplo queremos que a função imprima os valores das raízes e retorne o valor de delta para o código que chamou a função.

```
def eq2grau():
    a = float(input("Entre com o coef a: "))
    b = float(input("Entre com o coef b: "))
    c = float(input("Entre com o coef c: "))
    Delta = b**2-4*a*c
    x1 = (-b-Delta**0.5)/(2*a)
    x2 = (-b+Delta**0.5)/(2*a)
    print("Soluções: x1 = ", x1, "x2 = ", x2)
    return Delta

var = eq2grau()
print("O valor de delta é ", var)
```

# Retorno de funções

O mais interessante é que poderíamos retornar além do discriminante, as raízes:

```
def eq2grau():
    a = float(input("Entre com o coef a: "))
    b = float(input("Entre com o coef b: "))
    c = float(input("Entre com o coef c: "))
    Delta = b**2-4*a*c
    x1 = (-b-Delta**0.5)/(2*a)
    x2 = (-b+Delta**0.5)/(2*a)
    print("Soluções: x1 = ", x1, "x2 = ", x2)
    return x1, x2, Delta
r1, r2, delta = eq2grau()
print("O valor de delta é ", delta, ", x1 = ", r1, ",
      x2 = ", r2)
```

Observe que as variáveis retornadas da função não precisam ter o mesmo nome das variáveis fora da função.

Uma função que nada retorna na verdade retorna None.



# Retorno de funções

O mais interessante é que poderíamos retornar além do discriminante, as raízes:

```
def eq2grau():
    a = float(input("Entre com o coef a: "))
    b = float(input("Entre com o coef b: "))
    c = float(input("Entre com o coef c: "))
    Delta = b**2-4*a*c
    x1 = (-b-Delta**0.5)/(2*a)
    x2 = (-b+Delta**0.5)/(2*a)
    print("Soluções: x1 = ", x1, "x2 = ", x2)
    return x1, x2, Delta
r1, r2, delta = eq2grau()
print("O valor de delta é ", delta, ", x1 = ", r1, ",
      x2 = ", r2)
```

Observe que as variáveis retornadas da função não precisam ter o mesmo nome das variáveis fora da função.

Uma função que nada retorna na verdade retorna None.

# Retorno de funções

O mais interessante é que poderíamos retornar além do discriminante, as raízes:

```
def eq2grau():
    a = float(input("Entre com o coef a: "))
    b = float(input("Entre com o coef b: "))
    c = float(input("Entre com o coef c: "))
    Delta = b**2-4*a*c
    x1 = (-b-Delta**0.5)/(2*a)
    x2 = (-b+Delta**0.5)/(2*a)
    print("Soluções: x1 = ", x1, "x2 = ", x2)
    return x1, x2, Delta
r1, r2, delta = eq2grau()
print("O valor de delta é ", delta, ", x1 = ", r1, ",
      x2 = ", r2)
```

Observe que as variáveis retornadas da função não precisam ter o mesmo nome das variáveis fora da função.

Uma função que nada retorna na verdade retorna None.

## 1 Python é a linguagem para aprender a programar

- Python é bem simples
- Python é uma linguagem não tipada
- Comentários em Python
- Codificação dos caracteres usados

## 2 Entrada e saída básica de dados

- Entrada de dados
- Saída de dados
- Método format
- Exercícios

## 3 Funções - parte 1

- Guardando códigos para reuso
- Retorno de funções
- Argumentos simples para funções

# Argumentos simples para funções

Suponha agora que queiramos que a função receba os coeficientes e faça os mesmos cálculos de antes. Para isto passamos os valores para a função, e na chamada informamos os valores:

```
def eq2grau(a, b, c):  
    Delta = b**2-4*a*c  
    x1 = (-b-Delta**0.5)/(2*a)  
    x2 = (-b+Delta**0.5)/(2*a)  
    return x1, x2, Delta  
  
r1, r2, delta = eq2grau(1, 7, 10)  
print("delta = ", delta, ", x1 = ", r1, ", x2 = ", r2)
```

Observe que agora podemos chamar várias vezes a função e fazer separadamente o cálculo.

# Argumentos simples para funções

Suponha agora que queiramos que a função receba os coeficientes e faça os mesmos cálculos de antes. Para isto passamos os valores para a função, e na chamada informamos os valores:

```
def eq2grau(a, b, c):  
    Delta = b**2-4*a*c  
    x1 = (-b-Delta**0.5)/(2*a)  
    x2 = (-b+Delta**0.5)/(2*a)  
    return x1, x2, Delta  
  
r1, r2, delta = eq2grau(1, 7, 10)  
print("delta = ", delta, ", x1 = ", r1, ", x2 = ", r2)
```

Observe que agora podemos chamar várias vezes a função e fazer separadamente o cálculo.

# Argumentos simples para funções

Para elucidar melhor como uma função pode ser usada vamos introduzir variáveis pseudo-aleatórias em python.

Imagine o lançamento de um dado, qualquer uma das faces (de 1 a 6) pode ser obtida aleatoriamente num lançamento.

Podemos simular isto utilizando a função `randint` da biblioteca `random`.

Antes de usar a função `randint` devemos chamar a biblioteca `random`, que precisa estar instalada previamente. Isto é feito escrevendo a linha:

```
import random as rn
```

Agora podemos chamar a função:

```
a = rn.randint(2,20)
```

Isto criará uma variável `a` que receberá um aleatório entre 2 e 20 (inclusive).

Para ver o funcionamento da função escreva o script:

```
import random as rd
for _ in range(10):          #Laço de 10 repetições
    a = rd.randint(2,20)
    print(a)
```

# Argumentos simples para funções

Para elucidar melhor como uma função pode ser usada vamos introduzir variáveis pseudo-aleatórias em python.

Imagine o lançamento de um dado, qualquer uma das faces (de 1 a 6) pode ser obtida aleatoriamente num lançamento.

Podemos simular isto utilizando a função `randint` da biblioteca `random`.

Antes de usar a função `randint` devemos chamar a biblioteca `random`, que precisa estar instalada previamente. Isto é feito escrevendo a linha:

```
import random as rn
```

Agora podemos chamar a função:

```
a = rn.randint(2,20)
```

Isto criará uma variável `a` que receberá um aleatório entre 2 e 20 (inclusive).

Para ver o funcionamento da função escreva o script:

```
import random as rd
for _ in range(10):          #Laço de 10 repetições
    a = rd.randint(2,20)
    print(a)
```

# Argumentos simples para funções

Para elucidar melhor como uma função pode ser usada vamos introduzir variáveis pseudo-aleatórias em python.

Imagine o lançamento de um dado, qualquer uma das faces (de 1 a 6) pode ser obtida aleatoriamente num lançamento.

Podemos simular isto utilizando a função `randint` da biblioteca `random`.

Antes de usar a função `randint` devemos chamar a biblioteca `random`, que precisa estar instalada previamente. Isto é feito escrevendo a linha:

```
import random as rn
```

Agora podemos chamar a função:

```
a = rn.randint(2,20)
```

Isto criará uma variável `a` que receberá um aleatório entre 2 e 20 (inclusive).

Para ver o funcionamento da função escreva o script:

```
import random as rd
for _ in range(10):          #Laço de 10 repetições
    a = rd.randint(2,20)
    print(a)
```



# Argumentos simples para funções

Para elucidar melhor como uma função pode ser usada vamos introduzir variáveis pseudo-aleatórias em python.

Imagine o lançamento de um dado, qualquer uma das faces (de 1 a 6) pode ser obtida aleatoriamente num lançamento.

Podemos simular isto utilizando a função `randint` da biblioteca `random`.

Antes de usar a função `randint` devemos chamar a biblioteca `random`, que precisa estar instalada previamente. Isto é feito escrevendo a linha:

```
import random as rn
```

Agora podemos chamar a função:

```
a = rn.randint(2,20)
```

Isto criará uma variável `a` que receberá um aleatório entre 2 e 20 (inclusive).

Para ver o funcionamento da função escreva o script:

```
import random as rd
for _ in range(10):          #Laço de 10 repetições
    a = rd.randint(2,20)
    print(a)
```

# Argumentos simples para funções

Para elucidar melhor como uma função pode ser usada vamos introduzir variáveis pseudo-aleatórias em python.

Imagine o lançamento de um dado, qualquer uma das faces (de 1 a 6) pode ser obtida aleatoriamente num lançamento.

Podemos simular isto utilizando a função `randint` da biblioteca `random`.

Antes de usar a função `randint` devemos chamar a biblioteca `random`, que precisa estar instalada previamente. Isto é feito escrevendo a linha:

```
import random as rn
```

Agora podemos chamar a função:

```
a = rn.randint(2,20)
```

Isto criará uma variável `a` que receberá um aleatório entre 2 e 20 (inclusive).

Para ver o funcionamento da função escreva o script:

```
import random as rd
```

```
for _ in range(10):          #Laço de 10 repetições
```

```
    a = rd.randint(2,20)
```

```
    print(a)
```

# Argumentos simples para funções

Para elucidar melhor como uma função pode ser usada vamos introduzir variáveis pseudo-aleatórias em python.

Imagine o lançamento de um dado, qualquer uma das faces (de 1 a 6) pode ser obtida aleatoriamente num lançamento.

Podemos simular isto utilizando a função `randint` da biblioteca `random`.

Antes de usar a função `randint` devemos chamar a biblioteca `random`, que precisa estar instalada previamente. Isto é feito escrevendo a linha:

```
import random as rn
```

Agora podemos chamar a função:

```
a = rn.randint(2,20)
```

Isto criará uma variável `a` que receberá um aleatório entre 2 e 20 (inclusive).

Para ver o funcionamento da função escreva o script:

```
import random as rd
for _ in range(10):          #Laço de 10 repetições
    a = rd.randint(2,20)
    print(a)
```

# Argumentos simples para funções

Para elucidar melhor como uma função pode ser usada vamos introduzir variáveis pseudo-aleatórias em python.

Imagine o lançamento de um dado, qualquer uma das faces (de 1 a 6) pode ser obtida aleatoriamente num lançamento.

Podemos simular isto utilizando a função `randint` da biblioteca `random`.

Antes de usar a função `randint` devemos chamar a biblioteca `random`, que precisa estar instalada previamente. Isto é feito escrevendo a linha:

```
import random as rn
```

Agora podemos chamar a função:

```
a = rn.randint(2,20)
```

Isto criará uma variável `a` que receberá um aleatório entre 2 e 20 (inclusive).

Para ver o funcionamento da função escreva o script:

```
import random as rd
for _ in range(10):          #Laço de 10 repetições
    a = rd.randint(2,20)
    print(a)
```

# Argumentos simples para funções

Para elucidar melhor como uma função pode ser usada vamos introduzir variáveis pseudo-aleatórias em python.

Imagine o lançamento de um dado, qualquer uma das faces (de 1 a 6) pode ser obtida aleatoriamente num lançamento.

Podemos simular isto utilizando a função `randint` da biblioteca `random`.

Antes de usar a função `randint` devemos chamar a biblioteca `random`, que precisa estar instalada previamente. Isto é feito escrevendo a linha:

```
import random as rn
```

**Agora podemos chamar a função:**

```
a = rn.randint(2,20)
```

Isto criará uma variável `a` que receberá um aleatório entre 2 e 20 (inclusive).

Para ver o funcionamento da função escreva o script:

```
import random as rd
for _ in range(10):          #Laço de 10 repetições
    a = rd.randint(2,20)
    print(a)
```

# Argumentos simples para funções

Para elucidar melhor como uma função pode ser usada vamos introduzir variáveis pseudo-aleatórias em python.

Imagine o lançamento de um dado, qualquer uma das faces (de 1 a 6) pode ser obtida aleatoriamente num lançamento.

Podemos simular isto utilizando a função `randint` da biblioteca `random`.

Antes de usar a função `randint` devemos chamar a biblioteca `random`, que precisa estar instalada previamente. Isto é feito escrevendo a linha:

```
import random as rn
```

Agora podemos chamar a função:

```
a = rn.randint(2,20)
```

Isto criará uma variável `a` que receberá um aleatório entre 2 e 20 (inclusive).

Para ver o funcionamento da função escreva o script:

```
import random as rd
for _ in range(10):          #Laço de 10 repetições
    a = rd.randint(2,20)
    print(a)
```

# Argumentos simples para funções

Para elucidar melhor como uma função pode ser usada vamos introduzir variáveis pseudo-aleatórias em python.

Imagine o lançamento de um dado, qualquer uma das faces (de 1 a 6) pode ser obtida aleatoriamente num lançamento.

Podemos simular isto utilizando a função `randint` da biblioteca `random`.

Antes de usar a função `randint` devemos chamar a biblioteca `random`, que precisa estar instalada previamente. Isto é feito escrevendo a linha:

```
import random as rn
```

Agora podemos chamar a função:

```
a = rn.randint(2,20)
```

Isto criará uma variável `a` que receberá um aleatório entre 2 e 20 (inclusive).

Para ver o funcionamento da função escreva o script:

```
import random as rd
for _ in range(10):      #Laço de 10 repetições
    a = rd.randint(2,20)
    print(a)
```

# Argumentos simples para funções

Vamos usar a função `eq2grau(a, b, c)` definida anteriormente para mostrar como uma função pode ser chamada várias vezes para resolver um problema.

```
# -*- coding: utf-8 -*-
import random as rd

def eq2grau(a, b, c):
    Delta = b**2-4*a*c
    x1 = (-b-Delta**0.5)/(2*a)
    x2 = (-b+Delta**0.5)/(2*a)
    return x1, x2, Delta

for _ in range(10):
    a = rd.randint(1,10) #Gera aleatório entre 0 e 10.
    b = rd.randint(-5,5) #Gera aleatório entre -5 e 5.
    c = rd.randint(-2,7) #Gera aleatório entre -2 e 7.
    res = eq2grau(a, b, c)
    print("Coeficientes: {0}, {1}, {2}, soluções: {3:.2f},
          {4:.2f}.".format(a, b, c, res[0], res[1]))
```

**OBS:** Neste código colocamos o retorno de `eq2grau()` num vetor, e em breve estudaremos isto melhor.



# Argumentos simples para funções

Vamos usar a função `eq2grau(a, b, c)` definida anteriormente para mostrar como uma função pode ser chamada várias vezes para resolver um problema.

```
# -*- coding: utf-8 -*-
import random as rd

def eq2grau(a, b, c):
    Delta = b**2-4*a*c
    x1 = (-b-Delta**0.5)/(2*a)
    x2 = (-b+Delta**0.5)/(2*a)
    return x1, x2, Delta

for _ in range(10):
    a = rd.randint(1,10) #Gera aleatório entre 0 e 10.
    b = rd.randint(-5,5) #Gera aleatório entre -5 e 5.
    c = rd.randint(-2,7) #Gera aleatório entre -2 e 7.
    res = eq2grau(a, b, c)
    print("Coeficientes: {0}, {1}, {2}, soluções: {3:.2f},
          {4:.2f}".format(a, b, c, res[0], res[1]))
```

**OBS:** Neste código colocamos o retorno de `eq2grau()` num vetor, e em breve estudaremos isto melhor.

