PYTHON

Características

Características

- Simples

Características

- Simples
- Interpretada de alto nível

Características

- Simples
- Interpretada de alto nível
- Orientada a objetos

Princípios

Princípios

- Baseada em indentação.

Princípios

- Baseada em indentação.
- Extremamente flexível

Princípios

- Baseada em indentação.
- Extremamente flexível
- As regras são estabelecidas pela comunidade Python

Python 2 x Python 3

Python 2 x Python 3

- Em 2008, os desenvolvedores da Python Software Foundation decidiram criar a versão 3.

Python 2 x Python 3

- Em 2008, os desenvolvedores da Python Software Foundation decidiram criar a versão 3.
- As mudanças foram muitas e, por isso, Python 3 não é compatível com Python 2.

Python 2 x Python 3

- Em 2008, os desenvolvedores da Python Software Foundation decidiram criar a versão 3.
- As mudanças foram muitas e, por isso, Python 3 não é compatível com Python 2.
- A manutenção de Python 2.7 foi até 2020.

Python 2 x Python 3

- Em 2008, os desenvolvedores da Python Software Foundation decidiram criar a versão 3.
- As mudanças foram muitas e, por isso, Python 3 não é compatível com Python 2.
- A manutenção de Python 2.7 foi até 2020.
- Portanto, projeto novo, Python 3.

Interpretador

Interpretador

- IPython é o interpretador de Python.

Interpretador

- IPython é o interpretador de Python.
- É a ferramenta que **traduz** o código fonte em uma ação.

Interpretador

- IPython é o interpretador de Python.
- É a ferramenta que **traduz** o código fonte em uma ação.
- Permite ainda simples cálculos diretamente.

Interpretador

- IPython é o interpretador de Python.
- É a ferramenta que **traduz** o código fonte em uma ação.
- Permite ainda simples cálculos diretamente.

Operador de Atribuição

Operador de Atribuição

- Exemplo:

Operador de Atribuição

- Exemplo:

```
x = 10
```

Operador de Atribuição

- Exemplo:

```
x = 10
```

- O tipo da variável é **automaticamente** identificado por Python.

Operador de Atribuição

- Exemplo:

```
x = 10
```

- O tipo da variável é **automaticamente** identificado por Python.
- Os tipos primitivos de Python são:

Operador de Atribuição

- Exemplo:

```
x = 10
```

- O tipo da variável é **automaticamente** identificado por Python.
- Os tipos primitivos de Python são:
  - **`int`**: inteiro

Operador de Atribuição

- Exemplo:

```
x = 10
```

- O tipo da variável é **automaticamente** identificado por Python.
- Os tipos primitivos de Python são:
  - **`int`**: inteiro
  - **`float`**: números reais

Operador de Atribuição

- Exemplo:

```
x = 10
```

- O tipo da variável é **automaticamente** identificado por Python.
- Os tipos primitivos de Python são:
  - **`int`**: inteiro
  - **`float`**: números reais
  - **`str`**: string de caracteres

Operador de Atribuição

- Exemplo:

```
x = 10
```

- O tipo da variável é **automaticamente** identificado por Python.
- Os tipos primitivos de Python são:
  - **`int`**: inteiro
  - **`float`**: números reais
  - **`str`**: string de caracteres
  - **`bool`**: lógico (boleano)

Operador de Atribuição

- Exemplo:

```
x = 10
```

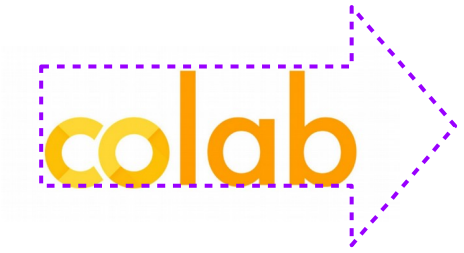
- O tipo da variável é **automaticamente** identificado por Python.
- Os tipos primitivos de Python são:
  - **`int`**: inteiro
  - **`float`**: números reais
  - **`str`**: string de caracteres
  - **`bool`**: lógico (boleano)
- O comando **`type`** é usado para identificar o tipo de uma variável.

Operador de Atribuição

- Exemplo:

```
x = 10
```

- O tipo da variável é **automaticamente** identificado por Python.
- Os tipos primitivos de Python são:
  - **`int`**: inteiro
  - **`float`**: números reais
  - **`str`**: string de caracteres
  - **`bool`**: lógico (boleano)
- O comando **`type`** é usado para identificar o tipo de uma variável.



Indentação

- Os limites das declarações e blocos de códigos são definidos pelo *layout* e não por símbolos específicos.
- Melhora na legibilidade e organização do código.
- Por convenção, nós usamos a indentação com quatro espaços.

Orientação a Objetos

Orientação a Objetos

- Python é uma linguagem orientada a objetos.

Orientação a Objetos

- Python é uma linguagem orientada a objetos.
- Em Python, **tudo** é objeto.

Orientação a Objetos

- Python é uma linguagem orientada a objetos.
- Em Python, **tudo** é objeto.
- Inclusive variáveis, funções e todas as estruturas são Python Object.

Orientação a Objetos

- Python é uma linguagem orientada a objetos.
- Em Python, **tudo** é objeto.
- Inclusive variáveis, funções e todas as estruturas são Python Object.
- Por exemplo, uma string é um objeto da classe `str`.

Orientação a Objetos

- Python é uma linguagem orientada a objetos.
- Em Python, **tudo** é objeto.
- Inclusive variáveis, funções e todas as estruturas são Python Object.
- Por exemplo, uma string é um objeto da classe `str`.
- Portanto, suas propriedades e métodos podem ser identificados através de um ponto (.).

Comentários

Comentários

- Necessários para melhorar a legibilidade do código e também para o compartilhamento de programas.

Comentários

- Necessários para melhorar a legibilidade do código e também para o compartilhamento de programas.
- Comentários em Python são delimitados por #.

Comentários

- Necessários para melhorar a legibilidade do código e também para o compartilhamento de programas.
- Comentários em Python são delimitados por #.
- As linhas com # não são executadas pelo interpretador de Python.

Comentários

- Necessários para melhorar a legibilidade do código e também para o compartilhamento de programas.
- Comentários em Python são delimitados por #.
- As linhas com # não são executadas pelo interpretador de Python.

Convenção de Nomes

Convenção de Nomes

- Python é *case sensitive*.

Convenção de Nomes

- Python é *case sensitive*.
- Convenções:

Convenção de Nomes

- Python é *case sensitive*.
- Convenções:
  - Variáveis sempre são minúsculas (*lowercase*) com separador de _ (*underline*).

Convenção de Nomes

- Python é *case sensitive*.
- Convenções:
  - Variáveis sempre são minúsculas (*lowercase*) com separador de _ (*underline*).
  - Funções sempre iniciam com letra maiúscula (*uppercase*) com separador de _ (*underline*).

Convenção de Nomes

- Python é *case sensitive*.
- Convenções:
  - Variáveis sempre são minúsculas (*lowercase*) com separador de _ (*underline*).
  - Funções sempre iniciam com letra maiúscula (*uppercase*) com separador de _ (*underline*).
  - Classes são escritas sem o separador com a primeira letra de cada nome em caixa alta.

Convenção de Nomes

- Python é *case sensitive*.
- Convenções:
  - Variáveis sempre são minúsculas (*lowercase*) com separador de _ (*underline*).
  - Funções sempre iniciam com letra maiúscula (*uppercase*) com separador de _ (*underline*).
  - Classes são escritas sem o separador com a primeira letra de cada nome em caixa alta.
  - Os pacotes são escritos em *lowercase.*

Convenção de Nomes

- Python é *case sensitive*.
- Convenções:
  - Variáveis sempre são minúsculas (*lowercase*) com separador de _ (*underline*).
  - Funções sempre iniciam com letra maiúscula (*uppercase*) com separador de _ (*underline*).
  - Classes são escritas sem o separador com a primeira letra de cada nome em caixa alta.
  - Os pacotes são escritos em *lowercase.*

Regras de Codificação

Regras de Codificação

- A *Python Software Foundation* estabelece uma série de recomendações sobre estilos de codificação.

Regras de Codificação

- A *Python Software Foundation* estabelece uma série de recomendações sobre estilos de codificação.
- https://www.python.org/dev/peps

Operadores Lógicos

- Se for necessário verificar se um objeto é de uma classe esperada, podemos fazer isto através do operador **is** ou **is not**.

Operadores Lógicos

- Se for necessário verificar se um objeto é de uma classe esperada, podemos fazer isto através do operador **is** ou **is not**.

Estruturas de Dados

Estruturas de Dados

- Python possui três importantes estruturas para armazenamento e manipulação de dados:

Estruturas de Dados

- Python possui três importantes estruturas para armazenamento e manipulação de dados:
  - Tuplas

Estruturas de Dados

- Python possui três importantes estruturas para armazenamento e manipulação de dados:
  - Tuplas
  - Listas

Estruturas de Dados

- Python possui três importantes estruturas para armazenamento e manipulação de dados:
  - Tuplas
  - Listas
  - Dicionários

Tuplas

Tuplas

- É uma estrutura que agrupa múltiplos objetos de maneira indexada.

Tuplas

- É uma estrutura que agrupa múltiplos objetos de maneira indexada.
- As tuplas são representadas através de parênteses.

Tuplas

- É uma estrutura que agrupa múltiplos objetos de maneira indexada.
- As tuplas são representadas através de

```
tupla1 = ('casa', 1, True)
```

Tuplas

- É uma estrutura que agrupa múltiplos objetos de maneira indexada.
- As tuplas são representadas através de

```
tupla1 = ('casa', 1, True)
```

- As tuplas possuem apenas uma dimensão.

Tuplas

- É uma estrutura que agrupa múltiplos objetos de maneira indexada.
- As tuplas são representadas através de

```
tupla1 = ('casa', 1, True)
```

- As tuplas possuem apenas uma dimensão.
- São objetos imutáveis.

Tuplas

- É uma estrutura que agrupa múltiplos objetos de maneira indexada.
- As tuplas são representadas através de parênteses.

```
tupla1 = ('casa', 1, True)
```

- As tuplas possuem apenas uma dimensão.
- São objetos imutáveis.
- As tuplas também podem ser criadas através da função tuple( ).

Tuplas

Tuplas

- Os valores são acessados através dos seus índices.

Tuplas

- Os valores são acessados através dos seus índices.

```
tupla5 = ('casa', 1, True)

tupla5[2]
```

Tuplas

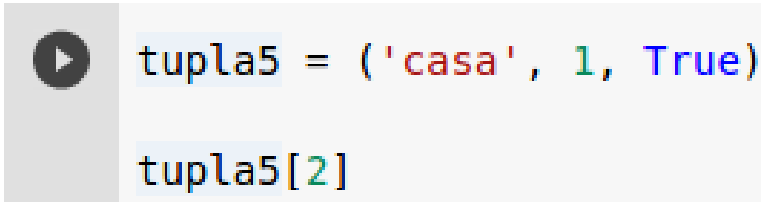
- Os valores são acessados através dos seus índices.

```
tupla5 = ('casa', 1, True)

tupla5[2]
```

- Tuplas podem ser interessantes porque exigem pouca memória.

Tuplas

- Os valores são acessados através dos seus índices.



```
tupla5 = ('casa', 1, True)

tupla5[2]
```

- Tuplas podem ser interessantes porque exigem pouca memória.
- São também usadas como saídas de funções.

Tuplas

- Os valores são acessados através dos seus índices.

```
tupla5 = ('casa', 1, True)

tupla5[2]
```

- Tuplas podem ser interessantes porque exigem pouca memória.
- São também usadas como saídas de funções.
- Tuplas são objetos.

Tuplas

- Os valores são acessados através dos seus índices.

```
tupla5 = ('casa', 1, True)

tupla5[2]
```

- Tuplas podem ser interessantes porque exigem pouca memória.
- São também usadas como saídas de funções.
- Tuplas são objetos.

```
tupla6 = ('casa', 10, 10, 10.1)

tupla6.count(10)
```

```
2
```

Tuplas

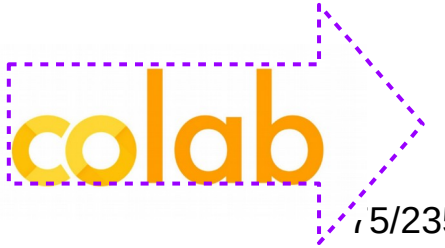
- Os valores são acessados através dos seus índices.

```
tupla5 = ('casa', 1, True)

tupla5[2]
```

- Tuplas podem ser interessantes porque exigem pouca memória.
- São também usadas como saídas de funções.
- Tuplas são objetos.

```
tupla6 = ('casa', 10, 10, 10.1)

tupla6.count(10)
```

```
2
```



Listas

Listas

- Estrutura que permite armazenar objetos de tipos distintos.

Listas

- Estrutura que permite armazenar objetos de tipos distintos.
- É uma estrutura modificável.

Listas

- Estrutura que permite armazenar objetos de tipos distintos.
- É uma estrutura modificável.
- Os valores são representados através de colchetes **[ ]**.

Listas

- Estrutura que permite armazenar objetos de tipos distintos.
- É uma estrutura modificável.
- Os valores são representados através de colchetes **[ ]**.
- Exemplo:

Listas

- Estrutura que permite armazenar objetos de tipos distintos.
- É uma estrutura modificável.
- Os valores são representados através de colchetes **[ ]**.
- Exemplo:

```
lista1 = [10, 3, True, 'casa']
```

Listas

Listas

- As listas também podem ser criadas através da função **list( )**.

Listas

- As listas também podem ser criadas através da função **list( )**.

- Essa função é comumente usada para converter outros tipos de objetos em listas.

Listas

- As listas também podem ser criadas através da função **list( )**.

```
lista2 = list(('10', 3, True))

lista2
```

```
['10', 3, True]
```

- Essa função é comumente usada para converter outros tipos de objetos em listas.

Listas

Listas

- Vasto conjunto de funções.

Listas

- Vasto conjunto de funções.
- **append( )**

Listas

- Vasto conjunto de funções.
- **append( )**
- **insert(i , valor)**

Listas

- Vasto conjunto de funções.
- **append( )**
- **insert(i , valor)**
- **pop (i)**

Listas

- Vasto conjunto de funções.
- **append( )**
- **insert(i , valor)**
- **pop (i)**
- **reverse( )**

Listas

- Vasto conjunto de funções.
- **append( )**
- **insert(i , valor)**
- **pop (i)**
- **reverse( )**
- **extend( )**

Listas

- Vasto conjunto de funções.
- **append( )**
- **insert(i , valor)**
- **pop (i)**
- **reverse( )**
- **extend( )**
- **index(valor)**

Listas

- Vasto conjunto de funções.
- **append( )**
- **insert(i , valor)**
- **pop (i)**
- **reverse( )**
- **extend( )**
- **index(valor)**
- **count(valor)**

Listas

- Vasto conjunto de funções.
- **append( )**
- **insert(i , valor)**
- **pop (i)**
- **reverse( )**
- **extend( )**
- **index(valor)**
- **count(valor)**
- **remove(valor)**

Listas

Listas

- Fatiamento de listas

Listas

- Fatiamento de listas
  - lista[posA:posB]

Listas

- Fatiamento de listas
  - lista[posA:posB]
- Uso de índices negativos

Listas

- Fatiamento de listas
  - lista[posA:posB]
- Uso de índices negativos
  - lista[-1]

Strings

Strings

- São consideradas listas de caracteres.

Strings

- São consideradas listas de caracteres.
- A codificação de strings em Python 3 é Unicode.

Strings

- São consideradas listas de caracteres.
- A codificação de strings em Python 3 é Unicode.
- A criação de uma variável do tipo string é entre aspas:

Strings

- São consideradas listas de caracteres.
- A codificação de strings em Python 3 é Unicode.
- A criação de uma variável do tipo string é entre aspas:

- Uma string é uma lista de caracteres.

Strings

- São consideradas listas de caracteres.
- A codificação de strings em Python 3 é Unicode.
- A criação de uma variável do tipo string é entre aspas:

- Uma string é uma lista de caracteres.
- As funções de listas podem ser usadas diretamente nas strings

Strings

- São consideradas listas de caracteres.
- A codificação de strings em Python 3 é Unicode.
- A criação de uma variável do tipo string é entre aspas:

```
string1 = 'Curso de Ciencia de Dados'
string2 = "Curso de Ciencia de Dados"
```

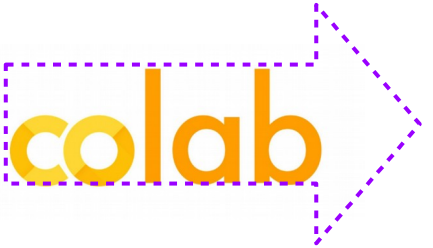
- Uma string é uma lista de caracteres.
- As funções de listas podem ser usadas diretamente nas strings

Strings

- São consideradas listas de caracteres.
- A codificação de strings em Python 3 é Unicode.
- A criação de uma variável do tipo string é entre aspas:

```
string1 = 'Curso de Ciencia de Dados'
string2 = "Curso de Ciencia de Dados"
```

- Uma string é uma lista de caracteres.
- As funções de listas podem ser usadas diretamente nas strings



Dicionários

Dicionários

- Tipo de estrutura bastante utilizada em diversas aplicações.

Dicionários

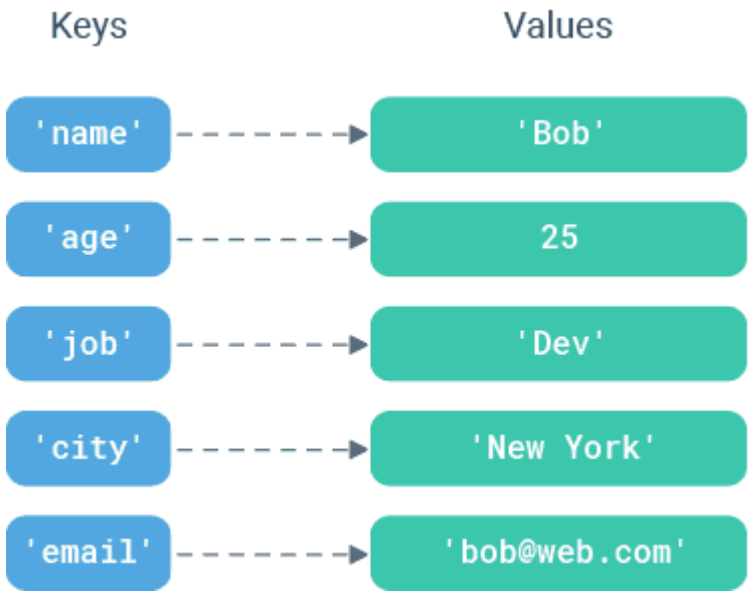
- Tipo de estrutura bastante utilizada em diversas aplicações.
- O acesso aos valores é através de uma chave, ao invés de usar um índice numérico (listas).

Dicionários

- Tipo de estrutura bastante utilizada em diversas aplicações.
- O acesso aos valores é através de uma chave, ao invés de usar um índice numérico (listas).
- Exemplo:

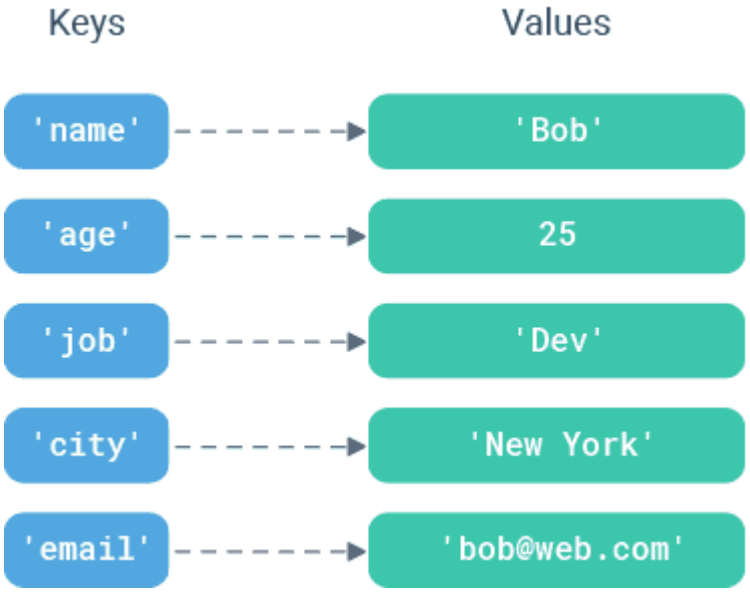
Dicionários

- Tipo de estrutura bastante utilizada em diversas aplicações.
- O acesso aos valores é através de uma chave, ao invés de usar um índice numérico (listas).
- Exemplo:



Dicionários

- Tipo de estrutura bastante utilizada em diversas aplicações.
- O acesso aos valores é através de uma chave, ao invés de usar um índice numérico (listas).
- Exemplo:



COMANDOS CONDICIONAIS

IF-ELSE

IF-ELSE

- Python usa o comando **if** para testar uma condição.

IF-ELSE

- Python usa o comando **if** para testar uma condição.
- Comandos simples e com sintaxe próxima de outras linguagens de programação.

IF-ELSE

- Python usa o comando **if** para testar uma condição.
- Comandos simples e com sintaxe próxima de outras linguagens de programação.
- A indentação deve ser mantida!

IF-ELSE

- Python usa o comando **if** para testar uma condição.
- Comandos simples e com sintaxe próxima de outras linguagens de programação.
- A indentação deve ser mantida!

LAÇOS DE REPETIÇÃO

Laços de Repetição

Laços de Repetição

- É comumente usado o comando **for**.

Laços de Repetição

- É comumente usado o comando **for**.
- É executado através de objetos iteráveis.

Laços de Repetição

- É comumente usado o comando **for**.
- É executado através de objetos iteráveis.
- Exemplo:

Laços de Repetição

- É comumente usado o comando **for**.
- É executado através de objetos iteráveis.
- Exemplo:

```
[37] for i in [10,20,30]:
       print(i)

     10
     20
     30
```

Range

Range

- Função usada para gerar uma sequência de números.

Range

- Função usada para gerar uma sequência de números.
- Exemplo 1:

Range

- Função usada para gerar uma sequência de números.
- Exemplo 1:

```
[40] print (list(range(5)))

     [0, 1, 2, 3, 4]
```

Range

- Função usada para gerar uma sequência de números.
- Exemplo 1:

```
[40] print (list(range(5)))

     [0, 1, 2, 3, 4]
```

- Exemplo 2:

Range

- Função usada para gerar uma sequência de números.
- Exemplo 1:

```
[40] print (list(range(5)))

 ⇨  [0, 1, 2, 3, 4]
```

- Exemplo 2:

```
[39] print (list(range(3,8)))

 ⇨  range(3, 8)
```

Range

- Função usada para gerar uma sequência de números.
- Exemplo 1:

```
[40] print (list(range(5)))

 ⇨  [0, 1, 2, 3, 4]
```

- Exemplo 2:

```
[39] print (list(range(3,8)))

 ⇨  range(3, 8)
```

- Exemplo 3:

Range

- Função usada para gerar uma sequência de números.
- Exemplo 1:

```
[40] print (list(range(5)))

     [0, 1, 2, 3, 4]
```

- Exemplo 2:

```
[39] print (list(range(3,8)))

     range(3, 8)
```

- Exemplo 3:

```
print (list(range(3,10,2)))

[3, 5, 7, 9]
```

Range

Range

- A função de range é comumente usada com **for**:

Range

- A função de range é comumente usada com **for**:

```
for i in range(5):
  print (i)
```

```
0
1
2
3
4
```

Enumerate

Enumerate

- A função retorna o índice (*index*) e o elemento de uma lista.

Enumerate

- A função retorna o índice (*index*) e o elemento de uma lista.
- Exemplo:

Enumerate

- A função retorna o índice (*index*) e o elemento de uma lista.
- Exemplo:

```
[44] for c in enumerate(cidades):
       print (c)

     (0, 'Manaus')
     (1, 'Itacoatiara')
     (2, 'Parintins')
     (3, 'Presidente Figueiredo')
```

Zip

Zip

- A função zip permite associar muitas listas e simultaneamente iterar através dos elementos destas listas.

While

While

- Python ainda possui o comando **while** como laço de repetição.

While

- Python ainda possui o comando **while** como laço de repetição.
- O controle do término do laço é responsabilidade do programador.

FUNÇÕES

Funções

Funções

- Um dos pontos fortes de Python é o poder de automatização de tarefas.

Funções

- Um dos pontos fortes de Python é o poder de automatização de tarefas.
- A forma mais simples de fazer isto é através das funções.

Funções

- Um dos pontos fortes de Python é o poder de automatização de tarefas.
- A forma mais simples de fazer isto é através das funções.
- Uma função é um objeto que aceita um parâmetro como entrada e então executa uma ação.

Funções

- Um dos pontos fortes de Python é o poder de automatização de tarefas.
- A forma mais simples de fazer isto é através das funções.
- Uma função é um objeto que aceita um parâmetro como entrada e então executa uma ação.
- Uma função usa a palavra reservada **def** e o nome da função.

Funções

- Um dos pontos fortes de Python é o poder de automatização de tarefas.
- A forma mais simples de fazer isto é através das funções.
- Uma função é um objeto que aceita um parâmetro como entrada e então executa uma ação.
- Uma função usa a palavra reservada **def** e o nome da função.
- Se a função retornar um valor, então deveremos usar o operador **return**.

Funções

Funções

- É possível ainda ter argumentos opcionais nas funções.

Funções

- É possível ainda ter argumentos opcionais nas funções.
- Para isto, é necessário definir os valores *default* destas funções.

Funções

- É possível ainda ter argumentos opcionais nas funções.
- Para isto, é necessário definir os valores *default* destas funções.
- Exemplo:

Funções

- É possível ainda ter argumentos opcionais nas funções.
- Para isto, é necessário definir os valores *default* destas funções.
- Exemplo:

```
def produto(a,b=10)
```

Funções

- É possível ainda passar listas e dicionários como parâmetros para funções.

Múltiplos Retornos

Múltiplos Retornos

- Uma função pode ainda retornar múltiplos valores.

Múltiplos Retornos

- Uma função pode ainda retornar múltiplos valores.
- Para isto, basta retornar os valores separados por vírgulas.

Funções Lambda

Funções Lambda

- São as chamadas funções anônimas.

Funções Lambda

- São as chamadas funções anônimas.
- Portanto, essas funções não têm nomes.

Funções Lambda

- São as chamadas funções anônimas.
- Portanto, essas funções não têm nomes.
- Elas são criadas em tempo de execução.

Funções Lambda

- São as chamadas funções anônimas.
- Portanto, essas funções não têm nomes.
- Elas são criadas em tempo de execução.
- São muito úteis na manipulação de dados em Ciência de Dados.

Funções Lambda

- São as chamadas funções anônimas.
- Portanto, essas funções não têm nomes.
- Elas são criadas em tempo de execução.
- São muito úteis na manipulação de dados em Ciência de Dados.
- Uma função anônima é criada através da palavra-reservada **lambda**.

Funções Lambda

- São as chamadas funções anônimas.
- Portanto, essas funções não têm nomes.
- Elas são criadas em tempo de execução.
- São muito úteis na manipulação de dados em Ciência de Dados.
- Uma função anônima é criada através da palavra-reservada **lambda**.

ORIENTAÇÃO A OBJETOS

Classes e Objetos

Classes e Objetos

- Python é uma linguagem orientada a objetos.

Classes e Objetos

- Python é uma linguagem orientada a objetos.
- Porém, é possível programar sem ter essa noção.

Classes e Objetos

- Python é uma linguagem orientada a objetos.
- Porém, é possível programar sem ter essa noção.
- A própria linguagem facilita essa abordagem.

Classes

Classes

- Os objetos são criados através das classes.

Classes

- Os objetos são criados através das classes.
- A programação em Python é através da manipulação de objetos.

Classes

- Os objetos são criados através das classes.
- A programação em Python é através da manipulação de objetos.
- Forma simples de criar um objeto a partir de uma classe:

Classes

- Os objetos são criados através das classes.
- A programação em Python é através da manipulação de objetos.
- Forma simples de criar um objeto a partir de uma classe:

objeto1 = classe1 (arg1)

Classes

- Os objetos são criados através das classes.
- A programação em Python é através da manipulação de objetos.
- Forma simples de criar um objeto a partir de uma classe:

  objeto1 = classe1 (arg1)

- A declaração abaixo criou uma variável ou objeto?

Classes

- Os objetos são criados através das classes.
- A programação em Python é através da manipulação de objetos.
- Forma simples de criar um objeto a partir de uma classe:

  objeto1 = classe1 (arg1)

- A declaração abaixo criou uma variável ou objeto?

```
valor = True
```

Classes

- Os objetos são criados através das classes.
- A programação em Python é através da manipulação de objetos.
- Forma simples de criar um objeto a partir de uma classe:

  objeto1 = classe1 (arg1)

- A declaração abaixo criou uma variável ou objeto?

```
valor = True
```

Como criar uma classe?

Como criar uma classe?

- Nós devemos usar a palavra-reservada **class**.
- O método construtor é representado através __init__.

Como criar uma classe?

- Nós devemos usar a palavra-reservada **class**.
- O método construtor é representado através __init__.

PACOTES E MÓDULOS

Introdução

Introdução

- Em Python, existem dois tipos de estruturas para organizar o código:

Introdução

- Em Python, existem dois tipos de estruturas para organizar o código:
  - Módulos

Introdução

- Em Python, existem dois tipos de estruturas para organizar o código:
  - Módulos
  - Pacotes

Introdução

- Em Python, existem dois tipos de estruturas para organizar o código:
  - Módulos
  - Pacotes
- Módulos são arquivos com a extensão .py em que funções e classes são armazenadas.

Introdução

- Em Python, existem dois tipos de estruturas para organizar o código:
  - Módulos
  - Pacotes
- Módulos são arquivos com a extensão .py em que funções e classes são armazenadas.
- Pacotes são estruturas de diretórios em que muitos módulos podem ser armazenados.

Instalação de Pacotes

Instalação de Pacotes

- É possível usar pacotes próprios ou desenvolvidos por terceiros.

Instalação de Pacotes

- É possível usar pacotes próprios ou desenvolvidos por terceiros.
- GitHub é um exemplo de repositório de pacotes.

Instalação de Pacotes

- É possível usar pacotes próprios ou desenvolvidos por terceiros.
- GitHub é um exemplo de repositório de pacotes.
- Instalação de pacotes usando Anaconda:

Instalação de Pacotes

- É possível usar pacotes próprios ou desenvolvidos por terceiros.
- GitHub é um exemplo de repositório de pacotes.
- Instalação de pacotes usando Anaconda:
  - **conda install numpy**

Instalação de Pacotes

- É possível usar pacotes próprios ou desenvolvidos por terceiros.
- GitHub é um exemplo de repositório de pacotes.
- Instalação de pacotes usando Anaconda:
  - **conda install numpy**
- Instalação de pacotes diretamente:

Instalação de Pacotes

- É possível usar pacotes próprios ou desenvolvidos por terceiros.
- GitHub é um exemplo de repositório de pacotes.
- Instalação de pacotes usando Anaconda:
  - **conda install numpy**
- Instalação de pacotes diretamente:
  - **pip install numpy**

Carregar Pacotes ou Módulos

Carregar Pacotes ou Módulos

- O comando import é usando para importar pacotes para a memória.

Carregar Pacotes ou Módulos

- O comando import é usando para importar pacotes para a memória.
- Exemplo:

Carregar Pacotes ou Módulos

- O comando import é usando para importar pacotes para a memória.
- Exemplo:
  - **import datetime**

Carregar Pacotes ou Módulos

- O comando import é usando para importar pacotes para a memória.
- Exemplo:
  - **import datetime**
- É possível ainda usar uma versão com nome mais curto do pacote.

Carregar Pacotes ou Módulos

- O comando import é usando para importar pacotes para a memória.
- Exemplo:
  - **import datetime**
- É possível ainda usar uma versão com nome mais curto do pacote.
- Exemplo:

Carregar Pacotes ou Módulos

- O comando import é usando para importar pacotes para a memória.
- Exemplo:
  - **import datetime**
- É possível ainda usar uma versão com nome mais curto do pacote.
- Exemplo:
  - **import numpy as np**

Carregar Pacotes ou Módulos

- O comando import é usando para importar pacotes para a memória.
- Exemplo:
  - **import datetime**
- É possível ainda usar uma versão com nome mais curto do pacote.
- Exemplo:
  - **import numpy as np**

Carregar Pacotes ou Módulos

Carregar Pacotes ou Módulos

- É possível ainda importar um pacote sem usar um prefixo.

Carregar Pacotes ou Módulos

- É possível ainda importar um pacote sem usar um prefixo.
- Exemplo:

Carregar Pacotes ou Módulos

- É possível ainda importar um pacote sem usar um prefixo.
- Exemplo:
  - **from pandas import ***

Carregar Pacotes ou Módulos

- É possível ainda importar um pacote sem usar um prefixo.
- Exemplo:
  - **from pandas import ***
- Esta abordagem deve ser evitada porque envolve muitos riscos de sobrescrever quando vários pacotes são carregados.

TRATAMENTO DE ERROS E EXCEÇÕES

Exceções

Exceções

- Python é considerada uma linguagem de programação flexível e pode facilmente levar a erros difíceis de serem encontrados (*debugados*).

Exceções

- Python é considerada uma linguagem de programação flexível e pode facilmente levar a erros difíceis de serem encontrados (*debugados*).
- Um programa robusto precisa tratar situações especiais.

Exceções

- Python é considerada uma linguagem de programação flexível e pode facilmente levar a erros difíceis de serem encontrados (*debugados*).
- Um programa robusto precisa tratar situações especiais.
- Por exemplo, divisão numérica cujo denominador é igual a zero.

EXPRESSÕES REGULARES

Expressões Regulares

Expressões Regulares

- São uma forma poderosa e rápida de realizar buscas em strings.

Expressões Regulares

- São uma forma poderosa e rápida de realizar buscas em strings.
- O pacote em Python que permite o uso de expressões regulares é **re**.

Expressões Regulares

- São uma forma poderosa e rápida de realizar buscas em strings.
- O pacote em Python que permite o uso de expressões regulares é **re**.
- Às vezes, expressões regulares podem ser difíceis de se compreender para os não acostumados.

DECORATORS

Decorators

Decorators

- São um tipo específico de função em Python que permite que o programador aplique restrições a qualquer uma função em Python.

Decorators

- São um tipo específico de função em Python que permite que o programador aplique restrições a qualquer uma função em Python.
- Um decorator toma alguma função, adiciona funcionalidades e retorna essa função.

Decorators

- São um tipo específico de função em Python que permite que o programador aplique restrições a qualquer uma função em Python.
- Um decorator toma alguma função, adiciona funcionalidades e retorna essa função.
- Às vezes, o uso desse recurso é também chamado de metaprogramação.

Funções

Funções

- Lembrar que tudo em Python são objetos.
- Funções também são objetos.
- Inclusive funções podem ser passadas como parâmetros.

Funções

- Lembrar que tudo em Python são objetos.
- Funções também são objetos.
- Inclusive funções podem ser passadas como parâmetros.

Decorators

Decorators

- Exemplo:

Decorators

- Exemplo:

```
def avancada(func):
    def inner():
        print("Função avançada - decorator")
        func()
    return inner


def basica():
    print("Função básica")
```

Decorators

- Exemplo:

```
def avancada(func):
    def inner():
        print("Função avançada - decorator")
        func()
    return inner


def basica():
    print("Função básica")
```

Decorators

- Exemplo:

DECORATOR

```
def avancada(func):
    def inner():
        print("Função avançada - decorator")
        func()
    return inner


def basica():
    print("Função básica")
```