

# Curso de Programação em Python

Módulo: Lógica de Programação

AROT-TEC



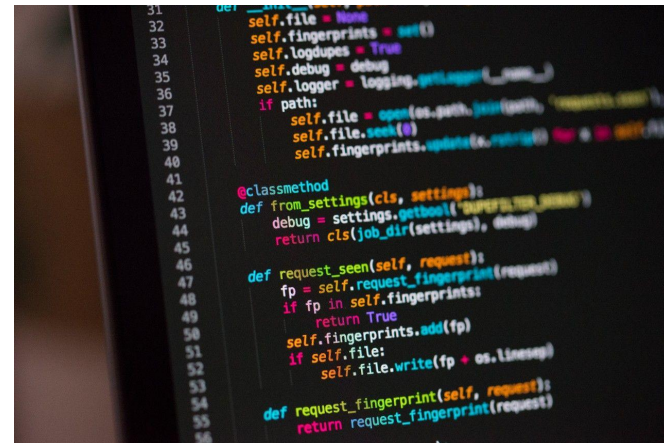
# Introdução ao Python

Python é uma linguagem de programação de alto nível, interpretada, imperativa, orientada a objetos, funcional, de tipagem dinâmica e forte. Foi lançada por Guido van Rossum . A elegância de sintaxe, a tipagem dinâmica e a natureza interpretativa tornam Python ideal para desenvolvimento rápido de aplicações em diversas áreas (Web, Machine Learning, Automação de tarefas) e plataformas (MacOS, Windows, Linux).



# Linguagens de programação

É uma forma de comunicação entre humanos e computadores que permite a criação de software e a automação de tarefas. Ela é composta por um conjunto de regras e sintaxe que os programadores usam para escrever código fonte. Esse código é traduzido por um compilador ou interpretador em instruções compreensíveis para o computador, permitindo que a máquina execute tarefas específicas de acordo com o programa criado



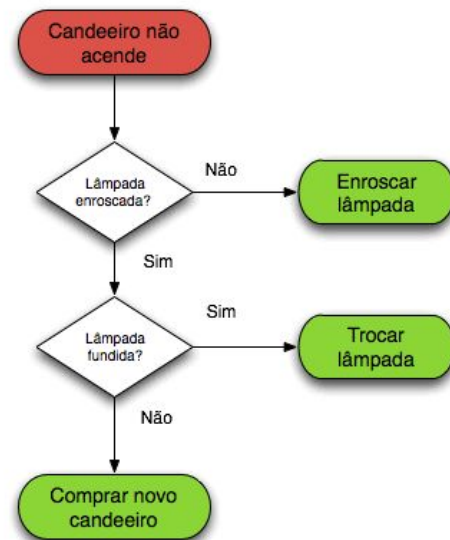
# Linguagens de baixo e alto nível

**Linguagens de Alto Nível:** São aquelas que se aproximam da forma como os seres humanos se comunicam. Exemplos incluem Python e Java. Elas oferecem simplicidade e legibilidade, permitindo que os programadores se concentrem nas funcionalidades e na lógica do programa

**Linguagens de Baixo Nível:** São mais facilmente compreendidas pelos sistemas computacionais (ou seja, pela máquina). Um exemplo é a linguagem Assembly, que oferece controle direto sobre o hardware do computador.

# Algoritmo

Algoritmo é uma sequência de instruções lógicas que descreve como resolver um problema ou realizar uma tarefa específica em programação. Essas instruções podem envolver cálculos matemáticos, manipulação de dados, decisões condicionais e repetições. O objetivo de um algoritmo é fornecer uma solução eficiente e correta para um determinado problema, independentemente da linguagem de programação utilizada.



# Instalação e Configuração

Durante a formação utilizaremos estas principais ferramentas:



**VS Code**



**Interpretador do  
Python**

# Comandos Básicos do Python

No Python temos dois principais comandos que são utilizados de forma cotidiana, estes são: **print** e **input**

**print:** é um comando que serve para enviar dados como saída

```
print('hello!')
```

**input:** é um comando que serve para receber dados vindos da entrada do usuário


```
nome = input('Digite o seu nome')
```

# Variáveis

Variável em qualquer linguagem de programação é definida como o espaço reservado na memória para armazenar dados.

Para declarar variáveis em Python devemos primeiramente informar o nome da variável, o símbolo de atribuição (=) e o valor que será armazenado.

**Ex:** idade=30



```
numero = 10  
nome = "Sebastião"  
altura = 1.66  
alto = True
```



# Uso de tipos de dados no Python

Lembra que o Python é de tipagem dinâmica isso permite que no Python a mesma variável possa receber valores de tipos diferentes ao longo da execução do código.

Muita gente acha que linguagem de tipagem dinâmica não possui tipos. Na verdade, ela possui tipos normalmente, só não precisam ser declaradas diretamente porque o interpretador assume de forma dinâmica de acordo o valor que assume. Existem 4 principais tipos de dados no Python:

**int, float, str, bool**

# Principais tipos de dados

**Inteiros (int):** representam números inteiros, como 1, 2, -3

**Números de ponto flutuante (float):** são utilizados para representar números decimais, como 3.14, -2.5

**Strings (str):** são sequências de caracteres, como “Olá, Mundo!”, “Python é incrível”. Strings em Python devem ser sempre delimitadas por aspas simples (") ou duplas ("")

**Booleano (bool):** representa dois valores: True (verdadeiro) ou False (falso)

## Conversão de tipos de dados

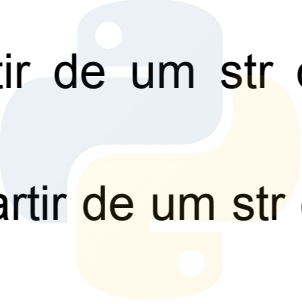
Pode haver momentos em que você queira converter um tipo para uma variável. Para conversão do tipo de uma variável existem uma função para cada tipo de dados.

**int()** cria um número inteiro a partir de um str ou float

**float()** cria um número decimal a partir de um str ou int

**str()** cria um string (sequência de caracteres) a partir de um str, int, float

**bool()** cria um valor booleano a partir de um str



```
decimal = float('5.7')
numero = int(decimal)
texto = str(numero)
solteiro = bool('True')
```

# Comentários

No Python é possível ignorar uma linha do nosso código para que o interpretador não leia utilizando o caráter: #

```
# Isto é um comentário  
print('Olá')
```

# Operadores

Operadores permitem realizar operações com variáveis ou valores constante. Existem 3 grupos de operadores.

- **Aritméticos (matemáticos)**

`+, -, *, /, %, //`

- **Relacionais (Comparação)**

`>, <, ==, >=, <=, !=`

- **Lógicos**

`and, or, not`



# Operadores Aritméticos (matemáticos)

Estes operadores apenas podem ser utilizados para operar números, e resultam também um valor numérico, seja int ou float.

**+: serve para somar**

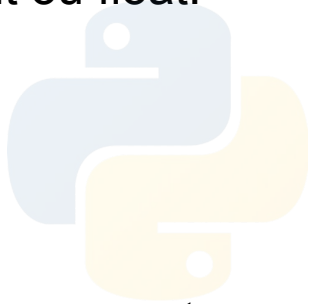
**-: serve para subtrair**

**\*: serve para multiplicar**

**/: serve para dividir**

**//: serve para dividir e ter apenas a parte inteira**

**?: serve para obter o resto da divisão**



```
a = 10
b=4
print(a+b) # 14
print(a-b) # 6
print(a*b) # 40
print(a/b) # 2.5
print(a//b) # 2
print(a%b) # 2
```

# Operadores Relacionais (Comparação)

Estes operadores apenas podem ser utilizados para operar números, e resultam em um valor booleano, ou seja True ou False.

**>: maior; <: menor; ==: igual; !=: diferente**

**>=: maior ou igual; <=: menor ou igual**

```
a = 10
b=4
print(a>b) # True
print(a<b) # False
print(a==b) # False
print(a!=b) # True
print(a>=b) # True
print(a<=b) # False
```

# Operadores Lógicos

Estes operadores apenas podem ser utilizados para operar valores booleanos, e resultam também um valor booleano, ou seja True ou False.

**and:** e ele só retorna True caso todos os valores operados sejam True;

**or:** ou ele só retorna True caso pelo menos um dos valores operados seja True;

**not:** não ele inverte o valor booleano de True para False ou de False para True

```
a = 10
b=4
c=5
# and
print(a>b and a > c) # True and True = True
print(a>b and a < c) # True and False = False

# or
print(a>b or a > c) # True and True = True
print(a>b or a < c) # True and False = True
print(a<=b or a == c) # False and False = False

#not
print(not a > b) # not True = False
print(not a < b) # not False = True
```



# Manipulação de Strings

Assim como as números e booleanos tem seus operadores e funções, as strings também possuem diversas funções. Abordaremos algumas agora e outra ao longo das lições:

**+** (função): junta dois strings

**upper:** torna todos os caracteres maiúsculos

**lower:** torna todos os caracteres minúsculos

**strip:** remove os espaços a mais

**find:** busca a posição de um ou mais caracteres

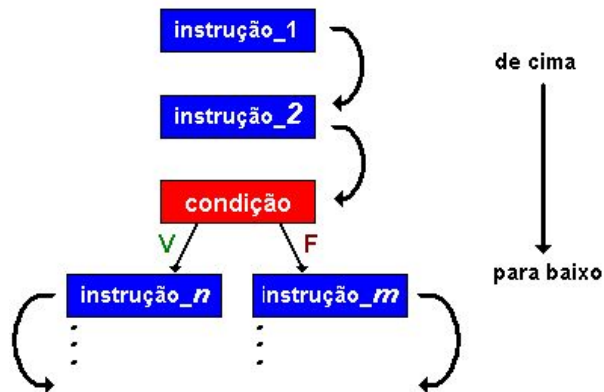
```
print('Lucas ' + 'Santos') # Lucas Santos
print('Lucas'.upper()) # LUCAS
print('Lucas'.lower()) # lucas
print('b'+ ' Lucas '.strip()+ 'b') # bLucasb
print('Lucas'.find('s')) # 5
```

# Estrutura Controle de Fluxo (Condicional)

Em muitas situações de um algoritmo ou programa precisamos analisar condições que especificar para onde vai o fluxo de execução, como a caso de verificar se indivíduo em uma campanha eleitoral maior de idade ou não.

Para isso temos **if**, **elif**, **else**

Seleção ou Condicional



## If e Elif e Else

**if** é usado para executar um bloco de código se uma condição especificada for verdadeira.

**elif** (abreviação de "else if") permite verificar condições adicionais se as condições anteriores não foram atendidas.

**else** é usado para executar um bloco de código se nenhuma das condições anteriores for verdadeira.

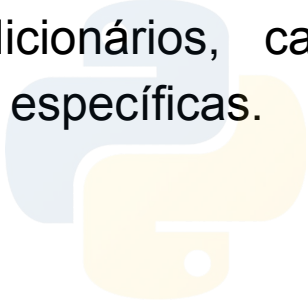
```
idade = 18
if idade > 18:
    print('É adulto.')
```

```
idade = 18
if idade >= 18:
    print('É adulto.')
elif idade >= 13:
    print('É adolescente')
elif idade >= 2:
    print('Criança')
```

```
idade = 18
if idade >= 18:
    print('É adulto.')
elif idade >= 13:
    print('É adolescente')
elif idade >= 2:
    print('Criança')
else:
    print('Bebê')
```

# Variáveis compostas

Em Python, as variáveis compostas são estruturas de dados poderosas que permitem armazenar coleções de elementos de maneira organizada e acessível. As três principais variáveis compostas em Python são listas, tuplas e dicionários, cada uma com suas características únicas e aplicações específicas.



# Listas

Listas são coleções ordenadas de itens, que podem ser de diferentes tipos (números, strings, outras listas, etc.). São representadas por colchetes [ ]. São mutáveis, ou seja, permitem acesso aos valores através de índices (indexs) que iniciam sempre de 0 e suportam operações como adicionar, remover e modificar elementos.

**append** - permite adicionar valores à lista

**remove** - permite remover valores a lista

**index** - permite obter índices de um valor na lista

```
nomes = ["Pedro", "Neto", "Dias"]
print(nomes) # ["Pedro", "Neto", "Dias"]
print(nomes[1]) # Neto
print(nomes[2]) # Dias
nomes.append('Francisco')
print(nomes) # ["Pedro", "Neto", "Dias", "Francisco"]
nomes.remove('Neto')
print(nomes) # ["Pedro", "Dias", "Francisco"]
nomes[0] = "Sukua" # Atualizando o valor no index 0
print(nomes.index("Dias")) # 1 ## Obtendo o index do valor Dias
print(nomes) # "Sukua", "Dias", "Francisco"]
```

# Tuplas

Tuplas são coleções ordenadas e imutáveis de itens. São representadas por parênteses ( ). Geralmente utilizadas para armazenar dados que não devem ser modificados. Acesso aos valores por índice, similar às listas.



```
tupla = (10, 20, 30)
print(tupla[0]) # 10
print(tupla[1]) # 20
print(tupla[2]) # 30
```

# Dicionários

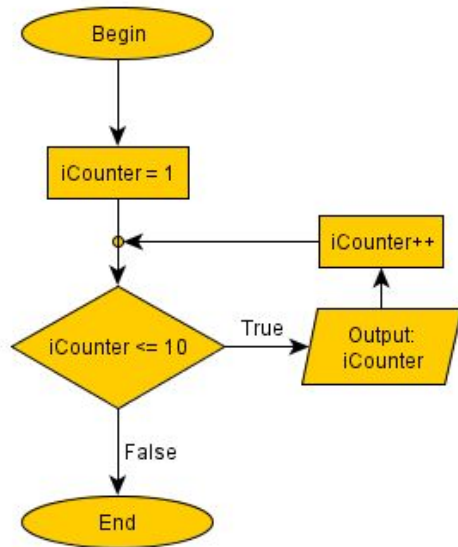
Dicionários são coleções desordenadas de pares chave-valor. São representadas por parênteses { }. Cada valor é acessado através de uma chave única.



```
peessoa = {  
    "nome": "Jairo Buto",  
    "idade": 18,  
    "curso": "Ciências da Computação"  
}  
print(peessoa["nome"]) # Jairo Buto  
print(peessoa["idade"]) # 18  
print(peessoa["curso"]) # Ciências da Computação
```

# Estruturas de Repetição

As estruturas de repetição em Python são recursos essenciais para automatizar tarefas que precisam ser executadas várias vezes, com base em condições específicas. Existem dois tipos principais de estruturas de repetição em Python: **while** e **for**





# While

O While executa um bloco de código repetidamente enquanto uma condição especificada for verdadeira. É ideal quando o número de repetições não é conhecido antecipadamente.

```
x = 0
while x < 5:
    print(x)
    x += 1
```

# For

O for é uma estrutura que permite iterar sobre valores de uma sequência ou iterável, executando um bloco de código repetidamente para cada elemento na sequência. É amplamente utilizado para percorrer listas, tuplas, strings, dicionários e outros objetos iteráveis em Python.

```
frutas = ["maçã", "banana", "laranja"]  
for fruta in frutas:  
    print(fruta)
```

# Funções

Em Python, as funções são blocos de código nomeados que executam tarefas específicas quando chamados. Elas permitem segmentar o código em partes menores, facilitando a organização e a reutilização.

Para criar uma função em Python, utilizamos a palavra-chave **def**, seguida do nome da função e, se necessário, parâmetros entre parênteses

```
def saudar():  
    print('Olá')  
saudar()
```

```
def soma(a, b):  
    return a + b  
  
resultado = soma(5, 3)  
print(resultado) # Saída: 8
```

# Escopo das Variáveis

As variáveis definidas dentro de uma função são chamadas de variáveis locais e só existem dentro do escopo da função.

Variáveis definidas fora da função são chamadas de variáveis globais e podem ser acessadas em todo o programa, incluindo dentro de funções

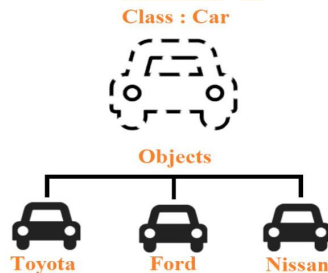
```
x = 10 # Variável global
def exemplo():
    y=5 # Variável local
    print(x) # Acesso à variável global é possível
    print (y)# Acesso à variável local é possível
exemplo()
print(x) # Saída: 10
print(y) # Erro: variável 'y' não está definida
```

# Classes

Lembra que definimos Python como uma linguagem orientada a objetos, a Programação Orientada a Objetos (POO) nada mais que um paradigma que permite modelar e estruturar o código em torno de objetos.

Uma classe é uma estrutura que define um novo tipo de dado, com atributos e métodos (funções) que representam suas características e comportamentos.

Um objeto é uma instância da classe, ou seja, é uma variável que contém os dados e comportamentos definidos pela classe.



# Criação de Classes

Para criar uma classe em Python, usamos a palavra-chave **class**, seguida do nome da classe.

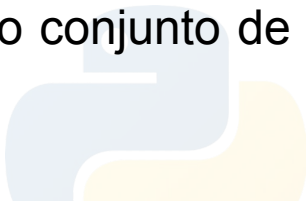
Os atributos são variáveis definidas dentro da classe e representam as características do objeto. Os métodos são funções definidas dentro da classe e representam o comportamento do objeto.

```
class Pessoa:
    def __init__(self, nome, idade):
        self.nome = nome
        self.idade = idade
    def saudacao(self):
        return f"Olá, meu nome é {self.nome} e tenho {self.idade} anos."
```

## Instanciando Objetos

Para criar um objeto da classe, basta chamar o nome da classe como se fosse uma função e passar os argumentos necessários para o construtor (método `__init__`).

Cada objeto criado terá seu próprio conjunto de atributos, independentes dos outros objetos.



```
peessoa1 = Pessoa('Pedro', 18)
peessoa2 = Pessoa('Marcos', 20)
```

# Módulos em Python

Em Python, um módulo é uma forma de organizar e reutilizar código. Um módulo é simplesmente um arquivo contendo definições de funções, classes, variáveis e/ou outros componentes que podem ser importados em outros programas Python. Isso ajuda a estruturar melhor o código, promove a reutilização e facilita a manutenção de grandes projetos de software.

Além de criar seus próprios módulos, Python oferece uma vasta biblioteca padrão de módulos built-in que você pode importar e usar diretamente, como `math`, `os`, `datetime`, entre outros. Além disso, existem milhares de módulos de terceiros disponíveis que podem ser instalados via `pip`.



# Criando um Módulo em Python

Para criar um módulo em Python, basta criar um arquivo .py contendo as definições que deseja encapsular. Por exemplo, considere o seguinte arquivo **slide\_modulo.py**. Você pode importar todas funções e variáveis ou específicas que estão definidas neste módulo em outro script Python.

```
#arquivo de modulo
```

```
def soma(a, b):  
    return a+b
```

```
PI = 3.14
```

```
# importando todas  
import slide_modulo
```

```
print(slide_modulo.soma(10, 5)) # 15  
print(slide_modulo.PI) # 3.14
```

```
# importando funções específicas  
from slide_modulo import soma  
print(soma(10, 5)) # 15
```

## Próximos Passos (Áreas de Atuação)

Após aprender sobre lógica de programação em qualquer linguagem que for, é fundamental decidir qual área vai decidir focar daqui em dia. Temos as seguintes áreas:

- **Web:** é o conjunto de processos envolvidos na criação de websites e aplicações que funcionam nos navegadores da web.
- **Mobile:** refere-se ao processo de criar e aprimorar aplicativos e sistemas para dispositivos móveis, como smartphones e tablets.

- **Games:** Desenvolvimento de jogos digitais e simulações interativas. Envolve gráficos 2D/3D, física computacional, inteligência artificial para jogabilidade, entre outros.
- **Ciência de Dados:** Utilização de técnicas estatísticas e algoritmos de aprendizado de máquina para extrair insights e conhecimentos de grandes volumes de dados.
- **Machine Learning e AI:** Desenvolvimento de sistemas que podem realizar tarefas que normalmente requerem inteligência humana. Usa técnicas como redes neurais, algoritmos de aprendizado supervisionado e não supervisionado para reconhecimento de padrões, processamento de linguagem natural, visão computacional

The Python logo, consisting of two interlocking snakes, one blue and one yellow, is positioned behind the text.

**OBRIGADO!**

**ARO-TEC**