

# Lógica de Programação e Orientação a Objetos

## Introdução ao Python

# Comandos Leia / Exiba

No português estruturado:

**Leia** → Receber um valor do usuário e armazena em uma variável.

**Exiba** → Exibir um texto ou valor de variável na tela.

Em Python:

```
nome = input("Digite o seu nome") # Corresponde ao Leia  
print("Seu nome: " + nome) # Corresponde ao Exiba
```

Obs: O caracter **#** será usado para denotar comentários e por isso o computador não interpretará como comandos.

# Comandos Leia / Exiba

Em Python:

```
nome = input("Digite o seu nome")  
print("Seu nome: " + nome)
```

Temos: o comando **input** exibirá na tela o texto **Digite o seu nome** e aguardará uma entrada do usuário. Essa entrada será armazenada na variável **nome**.

Supondo que o nome digitado tenha sido Bob. O comando **print** irá exibir na tela a mensagem **Seu nome: Bob**. Repare no sinal **+** que nesse caso indica uma concatenação e não uma soma.

# Programando

Escreva um programa que leia o nome de um aluno e escreva uma mensagem de boas vindas com o nome digitado: **Olá, <nome>. Seja bem-vindo!**

```
nome = input("Digite o seu nome")  
print("Ola, : " + nome + ". Seja bem-vindo!")
```

Apenas essas duas linhas de código são suficientes para ter um programa funcionando que cumpra o objetivo solicitado.

## str() / int()

As funções `str()` e `int()` são responsáveis pela conversão do tipo do dado, ou seja, altera o tipo da variável.

```
idade = 18  
print("Você tem " + idade + " anos de idade.")
```

Apesar de simples, o programa acima demonstra um problema ao ser executado:

**TypeError: Can't convert 'int' object to str implicitly**

Qual é o tipo da variável **idade**? Inteiro! Qual é o tipo dos textos "**Você tem** " e "**anos de idade.**" ? String!

## str() / int()

```
idade = 18  
print("Você tem " + idade + " anos de idades.")
```

**TypeError: Can't convert 'int' object to str implicitly**

Podemos somar um texto com um inteiro? Não. Nesse caso, precisamos converter a variável **idade** para uma string pois desejamos mostrá-la como texto na tela:

```
print("Você tem " + str(idade) + " anos de idades.")
```

## str() / int()

Escreva um programa que leia um número inteiro qualquer do usuário, multiplique por 2 e exiba o resultado na tela:

```
numero = input("Digite um numero: ")  
numero = numero * 2  
print("O resultado e " + str(numero))
```

Apesar do trecho acima ser válido, pode apresentar resultados divergentes e inconsistentes.

## str() / int()

Pra garantir que estamos lidando com os tipos certos do dado:

```
numero = int(input("Digite um numero: "))  
numero = numero * 2  
print("O resultado e " + str(numero))
```

Dessa forma conseguimos garantir que estamos lidando com um inteiro e a operação matemática seguinte produzirá resultados previsíveis e esperados.



# Operadores aritméticos

```
numero1 = 10
```

```
numero2 = 5
```

<code>numero1 + numero2</code>	<code># 10</code>	(Adição)
<code>numero1 - numero2</code>	<code># 5</code>	(Subtração)
<code>numero1 * numero2</code>	<code># 50</code>	(Multiplicação)
<code>numero1 / numero2</code>	<code># 2</code>	(Divisão)
<code>numero1 ** numero2</code>	<code># 100.000</code>	(Exponenciação)
<code>numero1 % numero2</code>	<code># 0</code>	(Resto da divisão)

# Operadores relacionais

```
numero1 = 10
```

```
numero2 = 5
```

<code>numero1 == numero2</code>	<code># False</code>	<code>(Igual a)</code>
<code>numero1 != numero2</code>	<code># True</code>	<code>(Diferente de)</code>
<code>numero1 &gt; numero2</code>	<code># True</code>	<code>(Maior que)</code>
<code>numero1 &lt; numero2</code>	<code># False</code>	<code>(Menor que)</code>
<code>numero1 &gt;= 10</code>	<code># True</code>	<code>(Maior ou igual a)</code>
<code>numero2 &lt;= 10</code>	<code># True</code>	<code>(Menor ou igual a)</code>

# Operadores lógicos

```
expressao1 = True  
expressao2 = False
```

```
expressao1 and expressao2  
expressao1 or expressao2  
not expressao1
```

```
# False      (E)  
# True       (Ou)  
# False      (NOT)
```

# Estrutura Se-Então

A estrutura de decisão Se-Então será representada da seguinte forma em Python:

```
numero = 10
if numero % 2 == 0:
    print("Numero par.")
else:
    print("Numero impar.")
```

Lê-se: Se o resto da divisão de **número** (10) por 2 for igual a 0, então exiba a mensagem **Numero par**. Caso contrário, exiba a mensagem **Numero impar**.

# Estrutura Enquanto-Faça

A estrutura de repetição Enquanto-Faça será representada da seguinte forma em Python:

```
numero = 10
while numero > 0:
    print(numero)
    numero = numero - 1
```

Lê-se: Enquanto a variável **numero** for maior que 0, exiba o valor da variável na tela e decremente 1 de **numero**.

Esse programa irá exibir os números de 1 a 10 em ordem decrescente.

# Funções

Um dos motivos de usarmos funções é evitar repetições de blocos de códigos iguais ou bem parecidos. As funções também ajudam a reaproveitar códigos já existentes.

Lembrando, devemos entender a estrutura de uma função uma vez que iremos usá-las e criar as nossas próprias com frequência.

## Nome

**Descrição do objetivo da função. Propósito da função.**

**Parâmetros (valores que a função recebe)**

**Retorno (valor que a função retorna)**

# Funções

## Nome

Descrição do objetivo da função. Propósito da função.

Parâmetros (valores que a função recebe)

Retorno (valor que a função retorna)

A estrutura de uma função que faça a soma entre dois números e exibe o resultado na tela:

## soma

Efetua a soma de dois números e mostra o resultado.

Parâmetros: dois números para serem somadas

Retorno: -

# Funções

```
def soma(numero1, numero2):  
    resultado = numero1 + numero2  
    print("A soma e: " + str(resultado))
```

A construção da função inicia pela palavra-chave **def** seguida do nome da função e a declaração dos parâmetros que a função deve receber. Quando finalizada, a função pode ser chamada a qualquer momento:

```
soma(10, 20)      # A soma e: 30
```

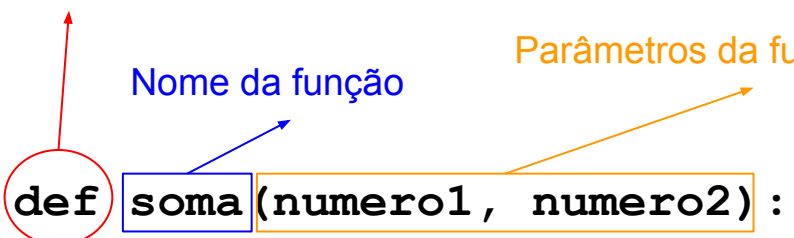


# Funções

Início da definição de uma função

Nome da função

Parâmetros da função



```
def soma(numero1, numero2):  
    resultado = numero1 + numero2  
    print("A soma e: " + str(resultado))
```

A definição da função não significa que ela será executada. Para que a função seja executada, é necessário fazer uma chamada no momento desejado.



# Retorno da função

Um peça não apareceu no exemplo anterior: o retorno da função. A função **soma** definida anteriormente mostra o resultado na tela porém não retorna um valor para o ponto a partir do qual foi chamado, nesse caso, o contexto principal.

```
def soma(numero1, numero2):  
    resultado = numero1 + numero2  
    print("A soma e: " + str(resultado))  
    # Não há um retorno definido nessa função
```

```
num1 = ...
```

```
num2 = ...
```

```
soma(num1, num2)
```

# Retorno da função

Podemos reescrever a função de modo que ela faça apenas a soma dos números ao invés de fazer a soma dos números e exibição do resultado:

**soma**

Efetua a soma de dois números **e retorna** o resultado.

Parâmetros: dois números para serem somadas

**Retorno: soma dos dois números**

```
def soma(numero1, numero2):  
    resultado = numero1 + numero2  
    return resultado
```

## Juntando as peças (2)

Escreva um programa que defina uma função que faça a soma de dois números. Os números devem ser informados pelo usuário e exibidos ao final.

```
def soma(numero1, numero2):  
    resultado = numero1 + numero2  
    return resultado  
  
num1 = int(input("Digite o primeiro numero: "))  
num2 = int(input("Digite o segundo numero: "))  
  
resposta = soma(num1, num2)  
print("A soma e: " + str(resposta))
```

## Juntando as peças (2)

```
def soma(numero1, numero2):  
    resultado = numero1 + numero2  
    return resultado
```

...



```
resposta = soma(num1, num2)  
print("A soma e: " + str(resposta))
```

O valor de retorno da função **soma** será armazenado na variável resposta.

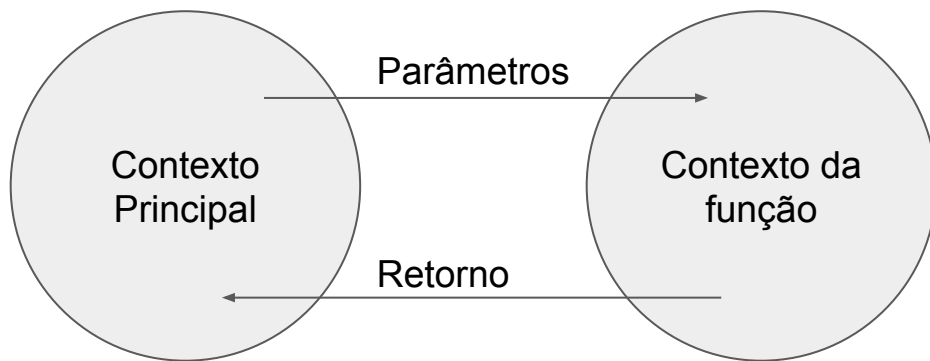
# Contexto das variáveis (Escopo)

Um conceito muito importante está relacionado ao contexto das variáveis, chamado de escopo. Como vimos em sala, o programa segue um **fluxo de execução principal** e quando uma função é chamada, acontece uma **troca de contexto**. Mundos diferentes.



# Contexto das variáveis (Escopo)

(1) Cada função tem seu próprio contexto. (2) A cada vez que a função finaliza e é chamada novamente, o contexto retorna ao estado inicial. (3) A forma que existe de trocar informações entre contextos é por parâmetros e através do retorno de uma função.





# Contexto das variáveis (Escopo)

O último exemplo de programa, se fosse visualizado dentro dos contextos, seria assim:

```
def soma(numero1, numero2):  
    resultado = numero1 + numero2  
    return resultado
```

Dentro da função a variável *numero1* terá o valor do primeiro parâmetro que foi passado: *num1*. E a variável *numero2* terá o valor do segundo parâmetros que foi passado: *num2*.

**Contexto da função**

```
num1 = int(input("Numero 1: "))  
num2 = int(input("Numero 2: "))  
  
r = soma(num1, num2)  
print("A soma e: " + str(r))
```

**Contexto da principal**

# Contexto das variáveis (Escopo)

Portanto, variáveis que são declaradas no contexto principal, não existirão no contexto da função. E variáveis declaradas no contexto da função, não existirão no contexto principal. **O meio de comunicação entre esses dois mundos é por parâmetros e retorno da função.**

# Estrutura Para-Até-Faça

A estrutura de repetição Para-Até-Faça será representada da seguinte forma:

```
for cont in range(10):  
    print("Número: " + str(cont))
```

Entende-se que a variável **cont** assumirá valores de 0 a 10, em ordem. Ou seja, no primeiro loop, a variável **cont** terá o valor 0. Na segunda, iteração assumirá o valor 1. E assim por diante, até o valor máximo 9.

Dessa forma, o resultado do código acima seria:

# Estrutura Para-Até-Faça

Número: 0

Número: 1

Número: 2

Número: 3

Número: 4

Número: 5

Número: 6

Número: 7

Número: 8

Número: 9

Lembrando que o comando **range** produz uma sequência de números que inicia em 0. Portanto **range(5)** produzirá os números 0, 1, 2, 3 e 4.

Dúvidas ?