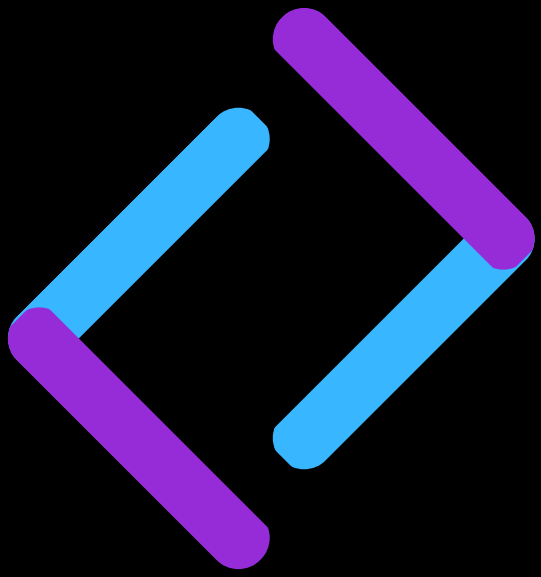


PETCode

Python

PET CC
Ciências Computacionais

PETCode

Indíce

O que é programação?

Programação é o processo de criar instruções que um computador pode seguir para executar uma tarefa. Essas instruções são escritas em uma linguagem de programação que o computador pode entender e executar.
A programação é como ensinar um computador a fazer algo, como resolver um problema matemático ou organizar dados. Os programadores escrevem códigos que são transformados em um programa executável pelo computador. O programa pode ser um aplicativo, um jogo ou até mesmo um site.

A programação envolve entender a lógica por trás de um problema, dividindo-o em pequenos passos e escrevendo um código que resolva cada etapa. Os programadores também precisam testar seus programas para garantir que eles funcionem corretamente e sem erros.

A programação é uma habilidade cada vez mais importante em muitas áreas de trabalho, desde ciência da computação até marketing digital. Aprender a programar pode ajudar a melhorar a eficiência, aumentar a produtividade e desenvolver novas soluções para problemas complexos.

O que é Python?

Python é uma linguagem de programação que foi criada para ser fácil de aprender e usar, com uma sintaxe clara e concisa. É uma linguagem de alto nível, o que significa que a programação em Python não requer o conhecimento de detalhes técnicos de baixo nível, como gerenciamento de memória, como é o caso de outras linguagens de programação.

Python é amplamente utilizado em muitas áreas diferentes, como ciência de dados, inteligência artificial, desenvolvimento web e automação de tarefas repetitivas. É uma linguagem de programação interpretada, o que significa que um programa Python pode ser executado diretamente no computador sem a necessidade de compilação prévia.

Uma das principais características do Python é sua ampla gama de bibliotecas de terceiros, que fornecem uma grande quantidade de funcionalidades e recursos adicionais para os programadores. Além disso, a comunidade de programadores Python é ativa e colaborativa, o que significa que há muitos recursos e suporte disponíveis para quem está aprendendo ou trabalhando com a linguagem.

Em resumo, Python é uma linguagem de programação popular e versátil, que é fácil de aprender e usar, e é amplamente utilizada em muitas áreas diferentes, tornando-se uma das linguagens mais populares para programação atualmente.

Instalação

Instalando o Python

Para instalar o Python em um computador, siga os seguintes passos:

1. Acesse o site oficial do Python em https://www.python.org/downloads/ e baixe a versão mais recente do Python para o seu sistema operacional. Certifique-se de escolher a versão correta para o seu sistema operacional (Windows, macOS, Linux, etc.) e a arquitetura (32 bits ou 64 bits).
2. Após o download ser concluído, execute o arquivo de instalação. Na maioria dos casos, o processo de instalação será intuitivo e você só precisará seguir as instruções na tela. Durante o processo de instalação, você pode optar por adicionar o Python ao PATH do sistema operacional, o que permite executar programas Python a partir do prompt de comando.
3. Após a instalação, você pode verificar se o Python foi instalado corretamente abrindo o prompt de comando (no Windows, pressione a tecla Win + R, digite "cmd" e pressione Enter) e digitando "python" no prompt. Se tudo estiver correto, o Python deve ser iniciado e você verá a versão instalada.
4. Para começar a programar em Python, você precisará de um editor de código. Existem muitas opções disponíveis, como Visual Studio Code, PyCharm, Sublime Text, entre outros.

Pronto! Agora você tem o Python instalado em seu computador e pode começar a aprender e programar em Python.

Instalando o Visual Studio Code

Para instalar o Visual Studio Code em um computador, siga os seguintes passos:

1. Acesse o site oficial do Visual Studio Code em https://code.visualstudio.com/ e baixe a versão mais recente do Visual Studio Code para o seu sistema operacional. Certifique-se de escolher a versão correta para o seu sistema operacional (Windows, macOS, Linux, etc.) e a arquitetura (32 bits ou 64 bits).
2. Após o download ser concluído, execute o arquivo de instalação. Na maioria dos casos, o processo de instalação será intuitivo e você só precisará seguir as instruções na tela.
3. Após a instalação, você pode iniciar o Visual Studio Code. Ao abrir o aplicativo pela primeira vez, você verá uma tela de boas-vindas e poderá escolher um tema, personalizar as configurações e instalar extensões para o Visual Studio Code.
4. Para começar a programar em Python no Visual Studio Code, você precisará instalar uma extensão para Python. Para isso, abra o Visual Studio Code e clique no ícone de extensões no painel lateral esquerdo. Em seguida, pesquise por "Python" e instale a extensão recomendada.

Pronto! Agora você tem o Visual Studio Code instalado em seu computador e pode começar a programar em Python. Lembre-se de que existem muitas outras extensões disponíveis para o Visual Studio Code, ele é um programa bem amplo e abrange várias linguagens de programação.

Conhecendo o Repl.it

Uma opção sem download e totalmente online e gratuito é a plataforma Repl.it, o único contra é que só funciona com uma rede conectada.
O repl.it é uma plataforma online que permite escrever, executar e compartilhar código em várias linguagens de programação, incluindo Python, Java, C++, JavaScript, entre outras. Se você é um iniciante em programação e deseja começar a usar o repl.it, siga os seguintes passos:

1. Acesse o site do repl.it em https://repl.it/ e crie uma conta gratuita. Você pode usar uma conta do Google ou do GitHub para facilitar o processo de registro.
2. Na página inicial do repl.it, clique no botão "New Repl" (Novo repl) para criar um novo projeto. Selecione a linguagem de programação que deseja usar, nomeie seu projeto e clique em "Create Repl" (Criar repl).
3. Você será direcionado para uma nova página, onde encontrará um editor de código e um console. Escreva o código que deseja executar no editor de código e clique no botão "Run" (Executar) para executar seu código no console.
4. Se você encontrar algum erro ao executar seu código, o console exibirá mensagens de erro que podem ajudá-lo a identificar e corrigir o problema. Além disso, você pode usar o console para interagir com seu código e testar diferentes entradas.
5. Para compartilhar seu projeto com outras pessoas, clique no botão "Share" (Compartilhar) na parte superior da página. Você pode gerar um link para compartilhar seu projeto ou convidar outras pessoas para colaborar em seu projeto.

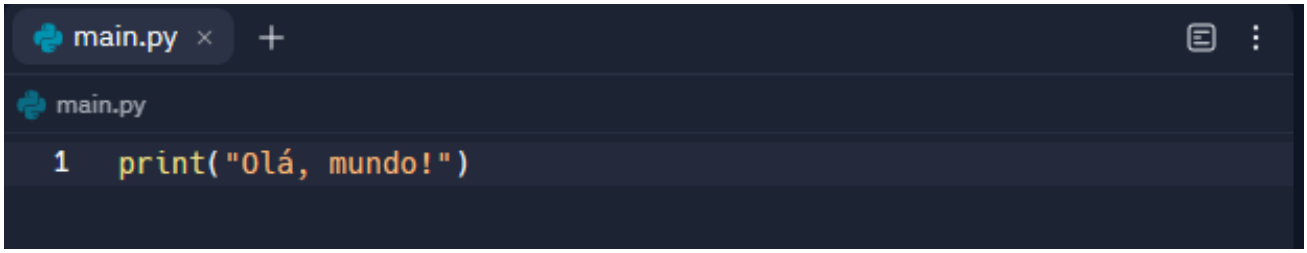
Pronto! Agora você pode começar a escrever e executar código em seu projeto no repl.it.

Comando Print

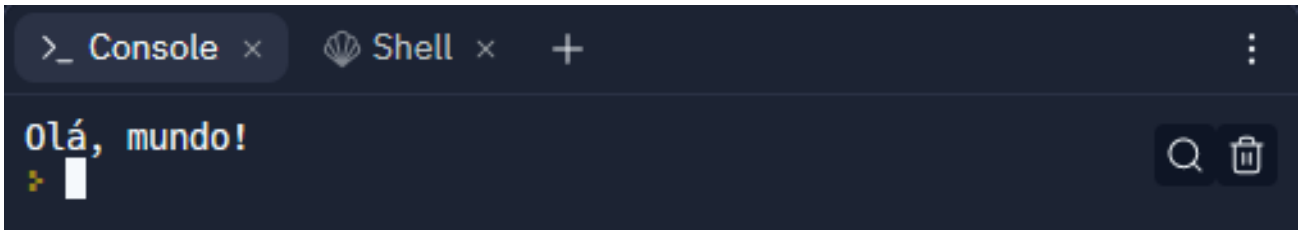
Print

O comando *print* em Python é a forma de saída de dados do programa, ou seja, é o comando que mostra mensagens na tela para o usuário. Essa mensagem pode ser uma saudação, variável, instrução, um texto informativo ou qualquer outro tipo de mensagem a ser passada.

Para ser utilizado, o comando *print* deve anteceder um par de parênteses e um par de aspas, duplas ou simples.

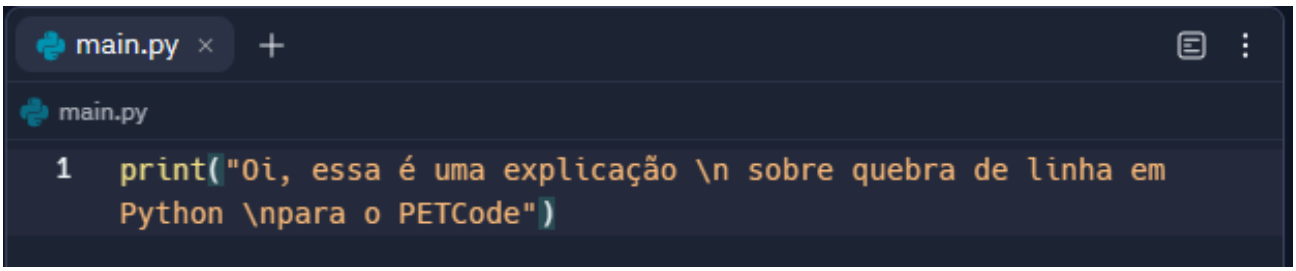


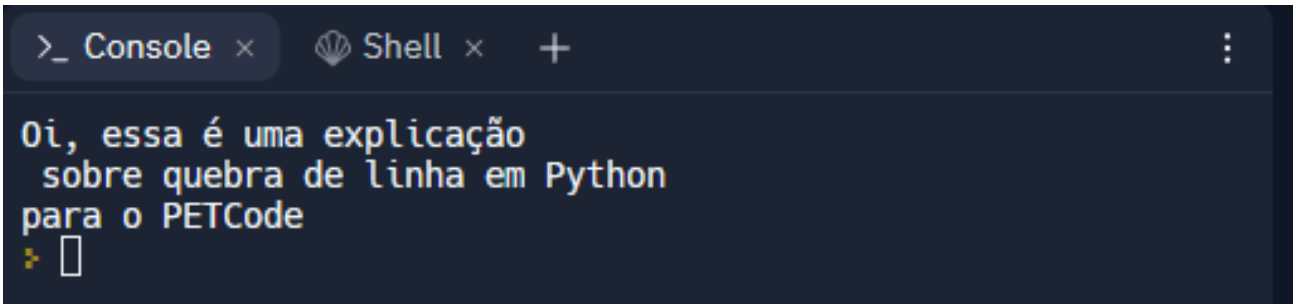
Resultando na seguinte mensagem:



A mensagem dentro das aspas é mostrada literalmente como está escrita, ou seja, não há correção de erros ortográficos ou sintaxe, bem como a diferenciação de elementos alfanuméricos ou caracteres especiais.

O comando de saída em Python tem quebra de linha automática, isso é, a cada comando dado, ele automaticamente começa uma nova linha. Caso queira quebrar linha no mesmo *print*, é necessário o comando `\n`, como mostra o exemplo a seguir:





É importante reparar que na segunda linha, por existir um espaço após o `\n`, a mensagem dela começa depois das outras duas.

Variáveis e Tipo de Dados

Variáveis

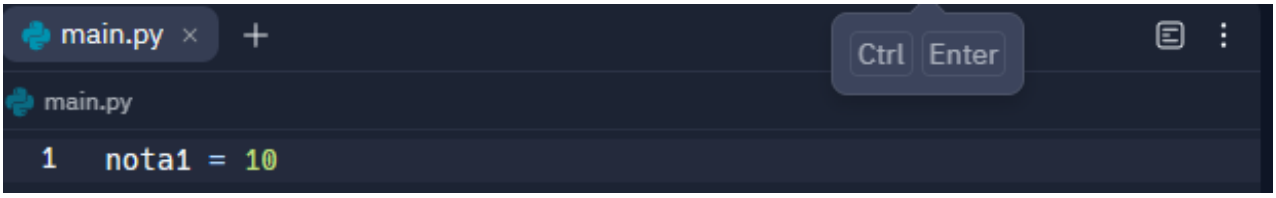
Uma variável é um espaço de armazenamento que contém um valor. Ela pode ser vista como uma caixa que guarda um valor específico, como um número ou um texto. Cada variável tem um nome que é usado para identificá-la no programa.

As variáveis são muito úteis na programação, pois permitem que os programadores armazenem e manipulem dados de uma maneira mais conveniente e organizada. Por exemplo, em um programa de cálculo de média de notas de alunos, podemos criar uma variável chamada "nota1" para armazenar a primeira nota e outra variável chamada "nota2" para armazenar a segunda nota.

Em Python, o nome de uma variável pode ser composto por letras, números e underscores (_), mas não pode começar com um número, caractere especial ou letras com acentuação gráfica. É importante escolher nomes significativos e descritivos para as variáveis, para que outras pessoas que leiam o código possam entender facilmente o que a variável representa. Por exemplo:

_esseNomeEhV4lido, ess3TAMBEM, #essenao, 3ssetambemnao, variáveltambemnao.

Para atribuir um valor a uma variável em Python, usamos o operador de atribuição "=" seguido pelo valor que queremos armazenar. Por exemplo, para atribuir o valor 10 à variável "nota1", usamos o seguinte comando:



Para printar o valor dessa variável, ela pode ser feita de diversas formas, mostraremos dois exemplos de como:

```
1  nota1 = 10
2  print("A nota tirada na prova foi", nota1)
```

Resultando em:

A segunda forma de fazer o mesmo print é com um "f" de formatação fora das aspas e o nome da variável dentro de {}, esse modo é mais fácil para casos que você necessite printar várias variáveis em um único print.

```
1  nota1 = 10
2  print(f"A nota tirada na prova foi {nota1}")
```

Resultando na mesma frase:

```
A nota tirada na prova foi 10
>
```

Tipos de Dados

Em Python, um tipo de dados é uma categoria de valores que podem ser armazenados em uma variável. Existem vários tipos de dados disponíveis em Python, incluindo:

- Inteiros (int): números inteiros, como 1, 2, 3, -4, -5, etc.
- Números de ponto flutuante (float): números com casas decimais, como 3.14, -1.5, 0.25, etc.
- Booleanos (bool): valores verdadeiros ou falsos, como True ou False.
- Strings (str): sequências de caracteres, como "Hello, world!", "Python", "123", etc.
- Listas (list): coleções ordenadas de valores, que podem incluir diferentes tipos de dados, como [1, 2, "three", 4.5].
- Tuplas (tuple): coleções ordenadas de valores, semelhantes às listas, mas são imutáveis, ou seja, não podem ser alteradas depois de criadas.
- Conjuntos (set): coleções não ordenadas de valores únicos, como {1, 2, 3}, {"hello", "world", "python"}, etc.
- Dicionários (dict): coleções de pares chave-valor, que permitem acessar os valores por meio de suas chaves, como {"nome": "João", "idade": 25, "cidade": "São Paulo"}.

É importante escolher o tipo de dados correto para a tarefa em questão, pois cada tipo tem suas próprias características e

limitações. Por exemplo, os números inteiros são adequados para cálculos matemáticos precisos, enquanto as strings são úteis para armazenar texto. As listas são usadas para armazenar coleções de valores que podem ser alterados, enquanto as tuplas são úteis para armazenar coleções de valores que não podem ser alterados. Os conjuntos são úteis para operações de conjunto, como união e interseção, enquanto os dicionários são usados para armazenar valores associados a chaves.

Para a ideia ficar mais mastigada, segue uma tabela:

Tipo	Descrição	Exemplo
int	Números inteiros	9, 13, -25
float	Números racionais. Com pontos flutuantes	0.5, 7.2, -4.5
str	Textos, palavras. Sempre dentro de aspas.	"casa", "Bernardo"
bool	Tipo booleano é basicamente verdadeiro ou falso.	"True" e "False"
list	Lista de valores. Cada item de uma lista corresponde a uma "casa"	[2, 4, 6]

Operadores

Operador de Atribuição (=)

O sinal de igualdade é um operador de atribuição no Python, ele é usado para definir o valor de uma variável. Seu uso é sempre na forma variável = valor (variável recebe valor). Esse valor pode variar de acordo com o tipo de dado. Por exemplo:

```python
x = 1
idade = 20
peso = 59.52
altura = 1.80
preco = 59.99
pi = 3.14159265359
nome = "Bernardo"
endereco = "Rua do Caju, 321"
saldo_bancario = -400.23
```

Operadores Aritméticos Básicos

Operadores Aritméticos Básicos são os necessários para solucionar operações em Python e eles, como na vida real, possuem um grau de prioridade. Para operações matemáticas mais complexas, existe uma biblioteca que pode ser importada, ela se chama *math*. https://docs.python.org/3/library/math.html

Operador	Operação	Exemplo
+	Adição	15 + 9 = 24
-	Subtração	25 - 3 - 7 = 15
/	Divisão	35 / 2 = 17.5
*	Multiplicação	7 * 3 = 21
//	Divisão Exata	35 // 2 = 17
**	Potência	4 ** 2 = 16
%	Resto	7 % 2 = 1

Comando Input

Input, entrada de dados, é uma função que permite que o usuário forneça dados para um programa enquanto ele está sendo executado. Esses dados são geralmente armazenados em uma variável para serem usados mais tarde no programa.

Por exemplo, imagine que você está escrevendo um programa que pede ao usuário para inserir o seu nome e depois o cumprimenta. Você pode usar a função input() para solicitar ao usuário que digite seu nome e armazenar o valor digitado em uma variável usando o sinal de igual (=). Em seguida, você pode usar a função print() para exibir uma mensagem de boas-vindas ao usuário, usando o valor armazenado na variável.

O input em Python permite que uma mensagem seja escrita na tela, como uma instrução.

Aqui está um exemplo de código simples que utiliza a função input():

```
1  nome = input("Qual o seu nome? ")
2  print(f"Olá, {nome}!")
```

Neste código, a variável "nome" recebe o nome digitado pelo usuário e usa essa string salva para poder cumprimentá-lo, como mostra a figura a seguir, onde o nome Sara foi digitado pelo usuário.

```
Qual o seu nome? Sara
Olá, Sara!
```

Além desse input mostrado, existe a possibilidade de colocarmos a especificação do dado antes do input, para que não haja conflito entre tipos de dados.

```python
nome = str(input("Qual o seu nome? "))
idade = int(input("Qual sua idade?"))
altura = float(input("Qual a sua altura em metros?"))
```

Aplicando o Conhecimento I

Agora que vimos print, input, tipo de dados, variáveis e operadores matemáticos, estamos aptos a testar esse conhecimento em pequenos programas.

O nosso primeiro programa será para fazer uma média entre três notas digitadas pelo usuário. Lembre-se que a nota de uma prova pode ser decimal.

```python
#Input referente a primeira nota.
nota1 = float(input("Digite aqui a nota da primeira avaliação: "))
#Input referente a segunda nota.
nota2 = float(input("Digite aqui a nota da segunda avaliação: "))
#Input referente a terceira nota
nota3 = float(input("Digite aqui a nota da terceira avaliação: "))

# Soma das notas em uma variável média.
media = (nota1 + nota2 + nota3) / 3

#Mensagem mostrando média ao usuário
print(f"A média do aluno é: {media}")
```

```
>_ Console × Shell × +
Digite aqui a nota da primeira avaliação: 8.5
Digite aqui a nota da segunda avaliação: 9
Digite aqui a nota da terceira avaliação: 5
A média do aluno é: 7.5
```

Para o segundo teste, resolveremos o seguinte problema:

Joaozinho quer calcular e mostrar a quantidade de litros de combustível gastos em uma viagem, ao utilizar um automóvel que faz 12 KM/L. Para isso, ele gostaria que você o auxiliasse através de um simples programa. Para efetuar o cálculo, deve-se fornecer o tempo gasto na viagem (em horas) e a velocidade média durante a mesma (em km/h). Assim, pode-se obter distância percorrida e, em seguida, calcular quantos litros seriam necessários. Mostre o valor com 3 casas decimais após o ponto.

Entrada (input)

O arquivo de entrada contém dois inteiros. O primeiro é o tempo gasto na viagem (em horas) e o segundo é a velocidade média durante a mesma (em km/h).

Saída (print)

Imprima a quantidade de litros necessária para realizar a viagem, com três dígitos após o ponto decimal

Observação importante: Como um print de um número float não possui limites para o número de casas decimais, teremos que usar um método no print para podermos limitar as casas decimais conforme nosso interesse.

Como estamos utilizando o método de print *f-string*, iremos adicionar *:.3f* logo após a variável, para termos três casas decimais. O algarismo indica quantas casas serão.

main.py

```python
# Primeiro input para descobrir o tempo gasto na viagem
tempo = float(input("Quanto tempo foi gasto na viagem? "))

# Segundo input para descobrir a velocidade média da viagem
velocidade = float(input("Qual foi a velocidade média? "))

# Cálculo da distância baseado na fórmula da velocidade media
# Vm = distancia / tempo

distancia = tempo * velocidade
```

```
11
12  # Agora que temos a distância total da viagem
13  # E com a informação de que o carro faz 12 km por litro de gasolina
14  # Podemos dividir a distância pelo consumo por litro, para descobrir o consumo
    total.
15
16  consumototal = distancia / 12
17
18  # Agora printaremos a solução, como foi especificado que o problema só quer
    três números das casas decimais, teremos que formartar o print.
19
20  print(f"O consumo total foi de {consumototal:.3f} litros.")
```

Estruturas de Seleção

O que é?

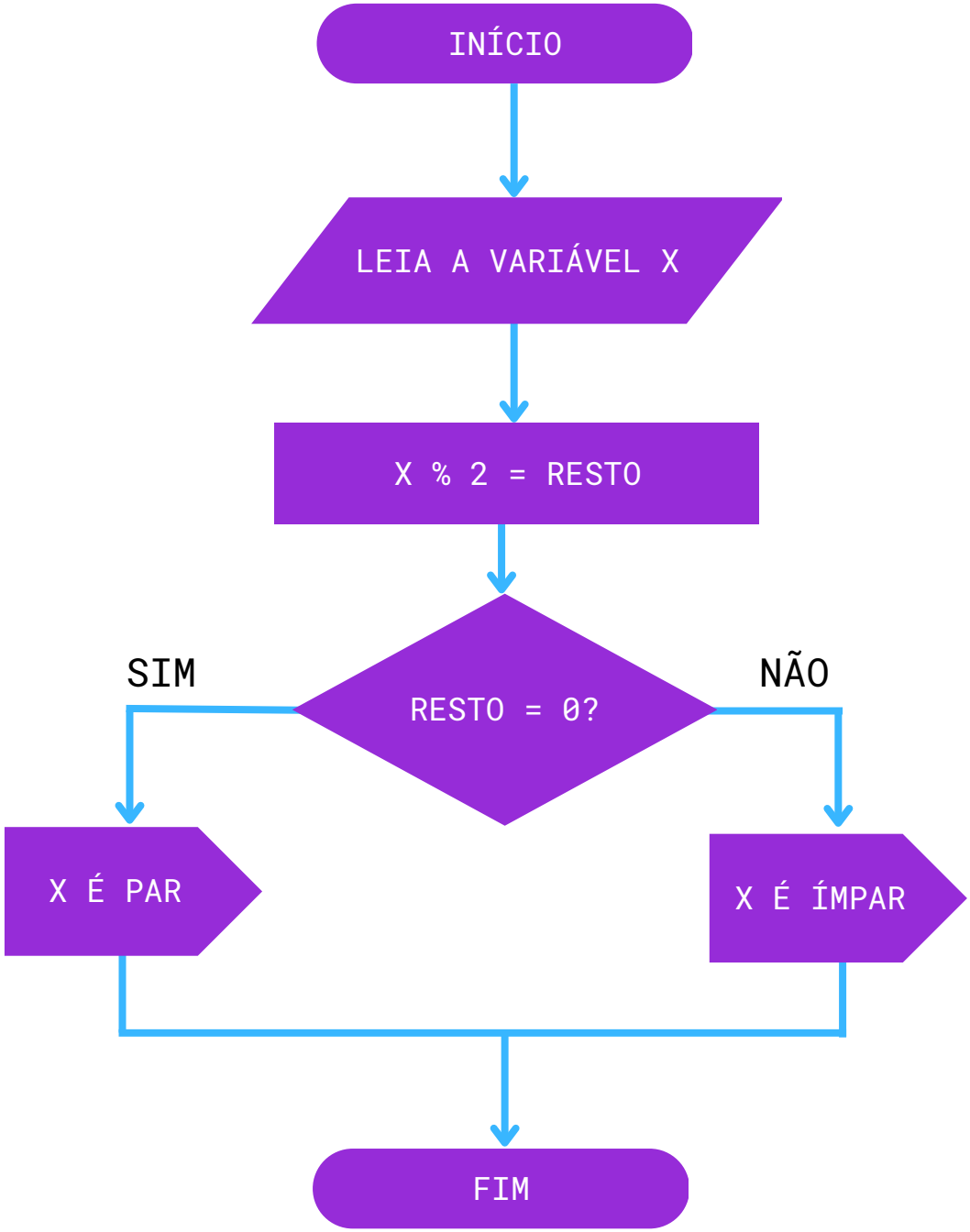
Uma estrutura de seleção é uma construção que permite selecionar um caminho específico de execução com base em uma condição ou um conjunto de condições. As estruturas de seleção são usadas para controlar o fluxo de execução de um programa, permitindo que diferentes seções de código sejam executadas dependendo das circunstâncias.

As estruturas de seleção em Python incluem principalmente as estruturas if, elif e else. O if é usado para testar uma única condição e executar um bloco de código se a condição for verdadeira. O elif é usado para testar múltiplas condições e executar um bloco de código correspondente à primeira condição verdadeira encontrada. O else é usado para executar um bloco de código se nenhuma das condições anteriores for verdadeira.

A ideia é basicamente "se algo acontecer, esse bloco de código rodará, senão, esse segundo bloco rodará.

If = Se, Else = Senão, Elif = Caso Se

Para deixarmos essa ideia menos abstrata, utilizaremos um fluxograma.



Há uma condição que pode ser simples ou composta, essa condição leva a diferentes caminhos.

OPERADORES DE COMPARAÇÃO

EXPRESSÃO	SIGNIFICADO	RESULTADO
a == b	Igual	True se a for igual a b
a != b	Diferente	True se a for diferente de b
a < b	Menor que	True se a for menor que b
a > b	Maior que	True se a for maior que b
a <= b	Menor ou igual	True se a for menor ou igual a b
a >= b	Maior ou igual	True se a for maior ou igual a b
a and b	E	True se a E b forem True
a or b	Ou	True se a OU b forem True
not a	Não	True se a for False

Exemplos de uso de operadores de comparação:

```
(x >= 1) and (x <= 10)  # X é um número entre 1 e 10
(x > 1) or (x < 10)     # X é maior que 1 ou menor que 10
x != 5                  # X é diferente de 5
x == 5                  # X é igual a 5
x > 1 and (x < 10 or x == 11) # X é maior que 1 e menor que 10 ou exatamente
                              # igual a 11
```

Lembrando que em cálculos ou para registrar variáveis é usado somente um sinal de igualdade, os dois sinais "==" servem apenas para estruturas de seleção.

COMANDO IF

O **if** em Python é uma estrutura condicional que permite ao programa fazer uma escolha baseada em uma condição. O **if** é usado para testar se uma condição é verdadeira ou falsa e executar o código apropriado em cada caso.

O **if** é uma das estruturas fundamentais da programação em Python e é amplamente utilizado em todos os tipos de programas. Com o **if**, você pode criar programas que tomam decisões e se adaptam a diferentes cenários, tornando-os mais flexíveis e úteis para os usuários.

Você pode usar operadores de comparação, como **>**, **<**, **>=**, **<=**, **==** e **!=**, para comparar valores e criar condições. O **if** pode ser usado sozinho ou em conjunto com outras estruturas de controle, como **elif** e **else**, para criar fluxos de execução mais complexos em um programa.

ATENTE-SE COM A IDENTAÇÃO.

```
1 if condição:
2     # o que acontece caso a condição seja verdadeira
```

COMANDO ELSE

O **else** é um comando usado para executar um bloco de exceção, caso nenhuma das alternativas anteriores tenha sido considerada verdadeira. Em outras palavras, o bloco de código associado ao **else** é executado somente se a condição do **if** for avaliada como **False**.

Cada condição pode ter somente um **else**.

```
if condição:
    # o que acontece caso a condição seja verdadeira

else:
    # o que acontece caso o if não seja verdadeiro
```

COMANDO ELIF

Elif é uma palavra-chave usada em conjunto com a estrutura de controle if para testar várias condições de forma encadeada. elif é uma abreviação de "else if", ou seja, "senão, se".

É mais fácil entendermos com um exemplo, então lá vai:

```
if condição1:
    # código a ser executado se a condição1 for verdadeira
elif condição2:
    # código a ser executado se a condição2 for verdadeira
elif condição3:
    # código a ser executado se a condição3 for verdadeira
else:
    # código a ser executado se todas as condições anteriores forem falsas

```

O código dentro do bloco if é executado se a condição1 for verdadeira. Se a condição1 for falsa, o programa testará a condição2. Se a condição2 for verdadeira, o código dentro do bloco elif correspondente será executado.

Se a condição2 for falsa, o programa testará a condição3. Se a condição3 for verdadeira, o código dentro do bloco **elif** correspondente será executado. Se a condição3 for falsa, o bloco **else** será executado.

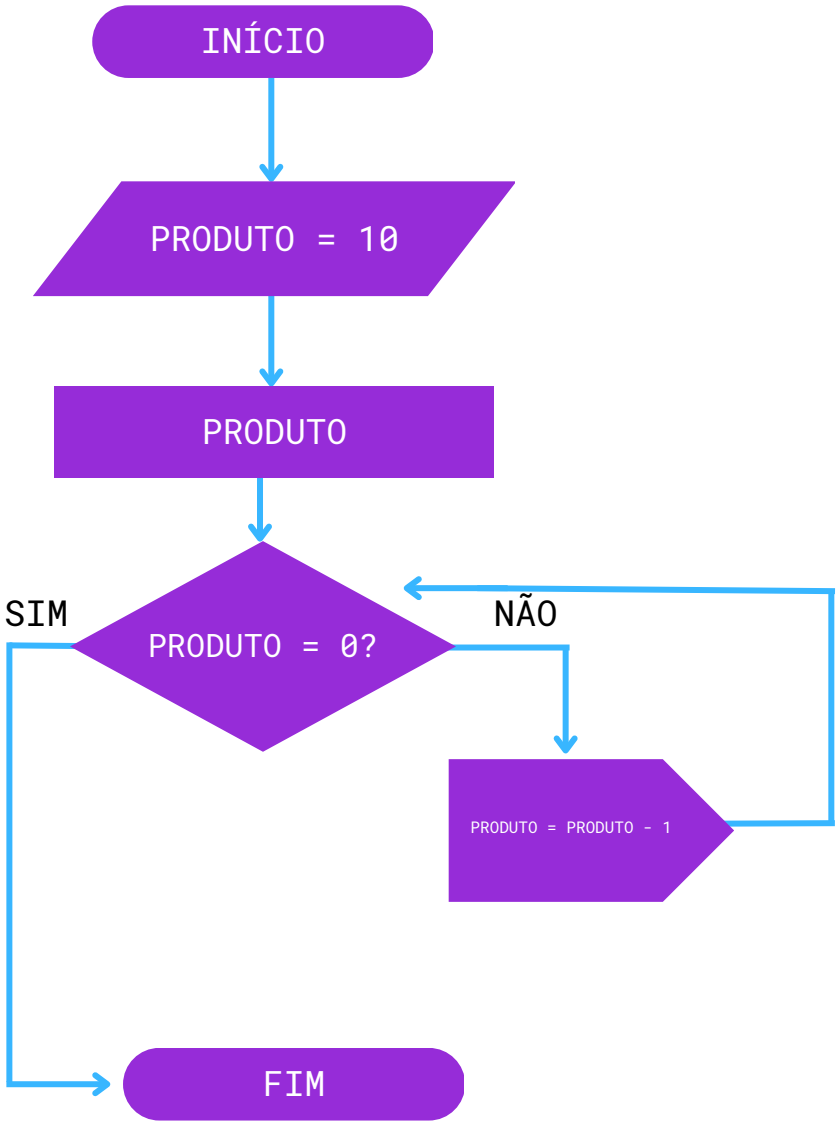
Agora usaremos o **if, elif e else** em um problema real.

```
idade = int(input("Digite sua idade: "))
#Pedido para o usuário digitar a sua idade

if idade < 18:
    print("Você é menor de idade")
#Caso a idade seja menor que 18 anos

elif idade >= 65:
    print("Você é idoso")
#Caso a idade seja maior ou igual a 65 anos

else:
    print("Você é adulto")
#Caso a idade seja entre 18 e 64 anos.
```

Estruturas de Repetição

O QUE É?

Uma estrutura de repetição (também conhecida como estrutura de laço ou loop) é uma construção que permite executar um bloco de código repetidamente enquanto uma determinada condição for verdadeira ou até que uma condição ocorra. As duas principais estruturas de repetição em Python são o loop **while** e o loop **for**.



Esse fluxograma mostra um problema onde o produto tem um valor 10, para que o programa seja finalizado, o produto precisa ser zero, então a cada repetição ele verifica o valor e subtrai um, caso não seja zero ainda.

LOOP WHILE

O loop **while** executa o bloco de código enquanto uma condição especificada for verdadeira. A sintaxe geral do loop while é a seguinte:

```
while condição:
    # código a ser repetido enquanto a condição for verdadeira
```

O código dentro do bloco **while** é executado repetidamente enquanto a condição especificada for verdadeira. Quando a condição se tornar falsa, a execução do loop é interrompida e o controle passa para a próxima instrução após o bloco **while**.

Por exemplo, imagine que queremos imprimir os números de 1 a 5 usando um loop **while**. Podemos fazer isso da seguinte forma:

```
i = 1
while i <= 5:
    print(i)
    i = i + 1
```

O resultado desse problema é:

```
>_ Console × Shell ×

1
2
3
4
5
```

O **i** é uma variável inicializada com valor 1. Enquanto o i for menor ou igual a 5, será impresso o valor de i na tela e logo após será somado 1 unidade ao valor de i.
E assim vai, até o i ter o valor 6 e sair do loop.

OBSERVAÇÃO: Caso não haja a soma no final do bloco while, ele irá imprimir o valor 1 infinitamente.

LOOP FOR

O **loop** for é uma estrutura de repetição que permite percorrer uma sequência de valores (como uma lista ou uma string) e executar um bloco de código para cada valor na sequência.

Em uma simples diferenciação, podemos dizer que o loop **for** é usado quando sabemos a quantidade de vezes que queremos repetir o laço, já o loop **while** pode ou não ter essa certeza de quantidade.

A sintaxe geral do **loop** for é a seguinte:

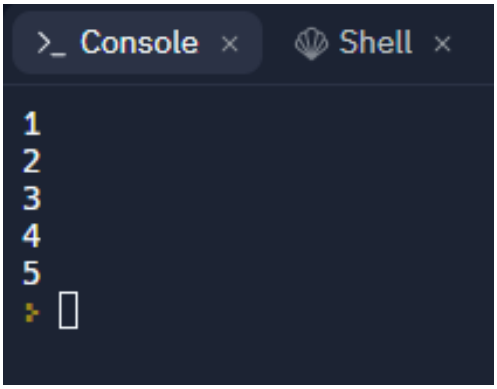
```
for valor in sequência:
    # código a ser repetido para cada valor na sequência
```

O código dentro do bloco **for** é executado uma vez para cada valor na sequência. O valor atual na sequência é armazenado na variável valor a cada iteração do loop.

Por exemplo, imagine que queremos imprimir os números de 1 a 5 usando um loop **for**. Podemos fazer isso da seguinte forma:

```
for i in range(1, 6):
    print(i)
```

Resultando no mesmo print do loop **while** visto anteriormente.



REPETIÇÕES ANINHADAS

Repetição aninhada em Python ocorre quando uma estrutura de repetição é usada dentro de outra estrutura de repetição.

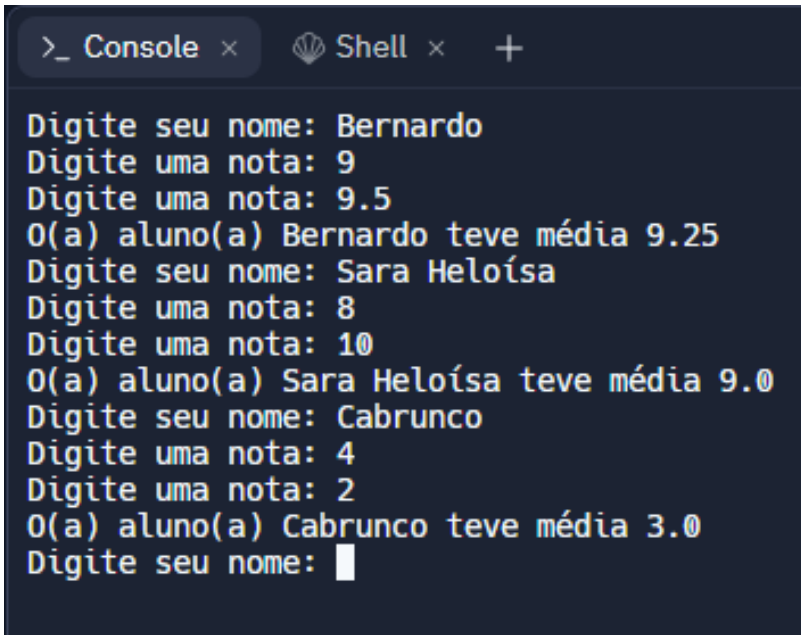
Por exemplo, imagine que você tem uma lista de listas (ou matriz), e você deseja percorrê-la inteiramente, linha por linha, item por item. Você pode usar um loop externo para percorrer cada linha e um loop interno para percorrer cada item em cada linha.

Como exemplo, usaremos um programa onde ele lerá duas notas de cada um dos 10 alunos de uma escola. Após cada duas leituras, ele fará o print da média dos alunos.

```
1  numero_de_Notas = 2
2  numero_de_Alunos = 10
3  for aluno in range(numero_de_Alunos):
4      nome = input("Digite seu nome: ")
5      soma = 0
6      for nota in range(numero_de_Notas):
7          valor = float(input("Digite uma nota: "))
8          soma = soma + valor
9          media = soma / numero_de_Notas
10     print(f"O(a) aluno(a) {nome} teve média {media}")
```

A cada primeiro loop, a variável soma vira zero, para no próximo loop ele receber um novo valor, dele mesmo mais o valor informado pelo usuário.

Como resultado temos:



LISTAS E STRINGS

O QUE É LISTA?

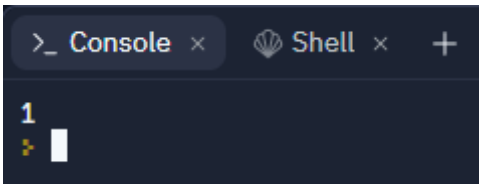
Uma lista é uma estrutura de dados que armazena uma coleção ordenada de elementos, que podem ser de qualquer tipo de dados, como **números**, **strings**, **objetos**, etc. As listas são uma das estruturas de dados mais flexíveis e comuns em Python, permitindo que você armazene e manipule facilmente dados em um programa.

As listas em Python são delimitadas por colchetes **[]** e os elementos são separados por vírgulas. Por exemplo, aqui está uma lista simples de números inteiros:

```
1   numeros = [1, 2, 3, 4, 5]
```

Você pode acessar um elemento específico da lista usando sua posição, ou índice, que começa com 0. Por exemplo, para acessar o primeiro elemento da lista acima, você usaria numeros[0], que retorna o valor 1.

```
1  numeros = [1, 2, 3, 4, 5]
2  print(numeros[0])
```

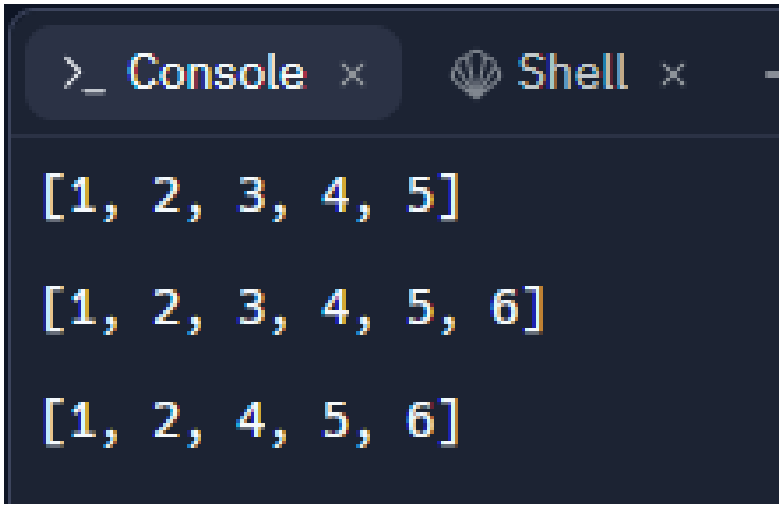


As listas também são mutáveis, o que significa que você pode adicionar, remover e modificar elementos em uma lista depois que ela é criada. Você pode adicionar elementos a uma lista usando o método **append()**, remover elementos usando o método **remove()** e modificar elementos acessando-os diretamente e atribuindo um novo valor.

Por exemplo:

```
1  numeros = [1, 2, 3, 4, 5]
2  print(numeros)
3  #Mostrar a lista atual de números
4
5  numeros.append(6) #Acrescenta o número 6 na lista
6  print(numeros)    # Mostra a lista editada, com novo número
7
8  numeros.remove(3) #Remove o número 3 da lista
9  print(numeros)    #Mostra a nova lista editada, sem o 3
```

Apresentando o resultado do código acima.



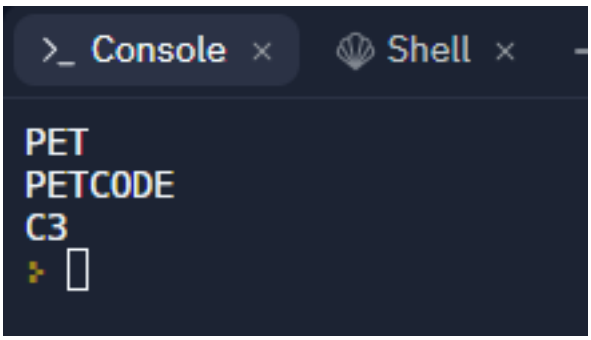
Uma lista pode ser registrada de duas formas, inicialmente com valores ou vazia, ambas podem ser mudadas conforme necessidade.

```
1  lista1 = []
2  lista2 = ["PET", "PETCODE", "C3"]
3  lista3 = [1, 2, 3, 4, 5]
4
```

Uma outra interação possível e frequente entre listas e loops é quando um loop roda entre uma lista e seus itens. Como mostrado a seguir:

```
1  lista = ["PET", "PETCODE", "C3"]
2  for i in lista:
3      print(i)
```

Isso retorna os seguintes prints:



O comando **insert** tem uma função parecida com a do **append**, mas a função **insert** insere um item em um local específico, já o **append** acrescenta no último item da lista.

```
1  GOT = ["Rey", "Leia", "Luke", "Yoda"]
2  print(GOT)
3  # lista feita com 4 nomes, do índice 0 ao 3
4
5  GOT.insert(2, "Chewbacca")
6  print(GOT)
7  # acrescentando um nome no índice 2, fazendo a lista
8  # ficar com 5 itens, do índice 0 ao 4
```

```
Resultado 1: ['Rey', 'Leia', 'Luke', 'Yoda']
Resultado 2: ['Rey', 'Leia', 'Chewbacca' ,'Luke', 'Yoda']
```

O QUE É STRING?

Uma **string** é uma sequência de caracteres. Ela é uma das estruturas de dados mais comuns e importantes em Python, usada para representar texto em um programa. **Strings** são definidas como um conjunto de caracteres em uma determinada ordem, incluindo letras, números, símbolos e espaços em branco.

As **strings** são criadas usando aspas simples (' ') ou duplas (" "). Por exemplo:

```python
nome = "Bernardo"
mensagem = "Olá, mundo!"
```

As **strings** em Python são imutáveis, o que significa que uma vez que você cria uma **string**, não é possível alterá-la. No entanto, você pode criar novas strings a partir de **strings** existentes usando operações como concatenação, fatiamento e formatação de **string**.

Algumas das operações comuns que você pode fazer com **strings** em Python incluem concatenar duas ou mais **strings** usando o operador +, extrair um caractere ou uma sequência de caracteres de uma **string** usando a indexação, obter o comprimento de uma string usando a função **len()**, e formatar uma **string** usando o método **format()**.

Concatenação:

```python
primeiro_nome = "Sara"
segundo_nome = "Heloísa"

nome_completo = primeiro_nome + " " + segundo_nome
# Concatenação das duas strings com um espaço entre elas
print(nome_completo)

```

```
Resultado: Sara Heloísa
```

Len:

```python
primeiro_nome = "Sara"
segundo_nome = "Heloísa"

nome_completo = primeiro_nome + " " + segundo_nome
comprimento = len(nome_completo)
# comprimento da string nome_completo
print(comprimento)
```

Resultado: 12

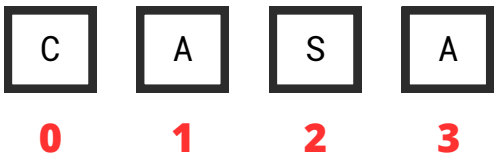
Format

```python
primeiro_nome = "Sara"
segundo_nome = "Heloísa"

nome_completo = primeiro_nome + " " + segundo_nome
mensagem = "Olá, {}".format(nome_completo)
# Acrescentando uma string existente em uma nova.
print(mensagem)
```

Resultado: Olá, Sara Heloísa

ÍNDICE DE STRINGS

Muito parecido com os índices de uma lista, que podem ser acessados separadamente pela posição deles, uma string também tem índice. Elas começam em zero, assim como as listas.



```python
primeiro_nome = "Sara"
segundo_nome = "Heloísa"

print(primeiro_nome[0])
# Mostrar a primeira letra da string
# primeiro_nome

print(segundo_nome[4])
# Mostrar a quinta letra da string
# segundo_nome

```

```
Resultado: S
           í
```

FUNÇÕES

O QUE É UMA FUNÇÃO?

Em programação, uma função é um conjunto de instruções que realizam uma tarefa específica. É como uma pequena sub-rotina ou mini-programa que pode ser chamado de outras partes do seu código. As funções são úteis porque permitem que você reutilize o mesmo bloco de código em diferentes partes do seu programa sem precisar reescrevê-lo.

O uso de funções em um código deixa ele mais limpo, mais fácil de tratar bugs e também permite a reutilização de código.

Existem algumas funções nativas do Python, funções que já vimos aqui nessa apostila, como a função **Print**, função **Input**, função **Len**

FUNÇÃO EM PYTHON

Em Python, você pode criar suas próprias funções definindo um bloco de código que executa uma tarefa específica e dá a ele um nome. As funções em Python são definidas usando a palavra-chave "**def**", seguida pelo nome da função e seus parâmetros, se houver, entre parênteses. O bloco de código da função deve ser indentado para indicar que é parte da função.

IDENTAÇÃO: def nomedafunção():

```
def cumprimento():
    print("Oi, prazer te conhecer.")
```

Para utilizar a função, deverá ser da seguinte forma:

USO: nomedafunção()

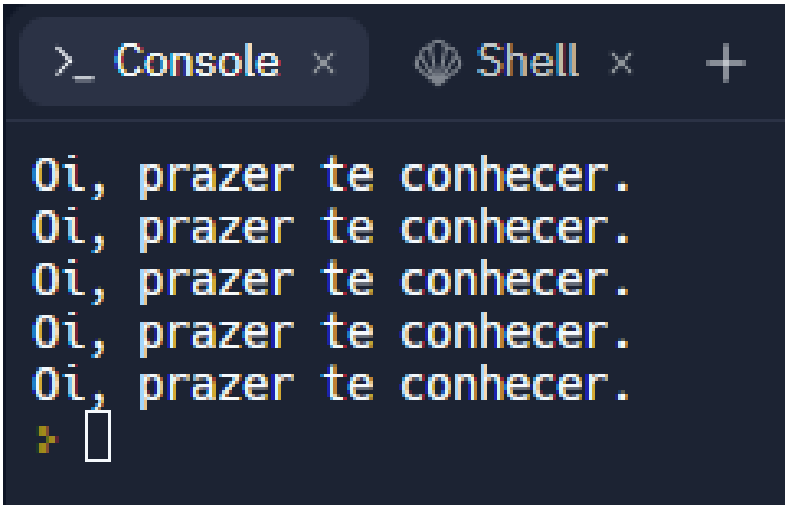
```
def cumprimento():
    print("Oi, prazer te conhecer.")

cumprimento()
```

Resultado: Oi, prazer te conhecer.

Uma utilização para uma função pode ser em um loop, como por exemplo:

```
def cumprimento():
    print("Oi, prazer te conhecer.")

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

Resultando no print a seguir:



ARGUMENTOS E RETORNOS

Argumentos e retornos de uma função são características de uso dela. Um argumento é uma informação que é passada para uma função durante sua chamada, e um retorno é o valor que uma função retorna após executar suas instruções.

Uma função pode ter nenhum, um ou quantos argumentos forem necessário.

O retorno é só um por função, mas ele pode retornar inúmeros dados para a função.

Para um melhor entendimento, usaremos primeiro serão mostradas e logo depois exemplificadas:

ARGUMENTOS

```
def nomedafunção(arg):
```

```
def nomedafunção(arg1, arg2):
```

RETORNO

```
def nomedafunção(arg):
    return dado
```

Agora veremos um exemplo de uma função de soma de dois números:

```python
def soma(x, y):
    resultado = x + y
    return resultado

```

Para utilizar essa função com dois números indicados pelo usuário, faremos assim:

main.py

```python
def soma(x, y):
    resultado = x + y
    return resultado

n1 = int(input("Digite um número: "))
n2 = int(input("Digite outro número: "))
conta = soma(n1, n2)
print(conta)

```

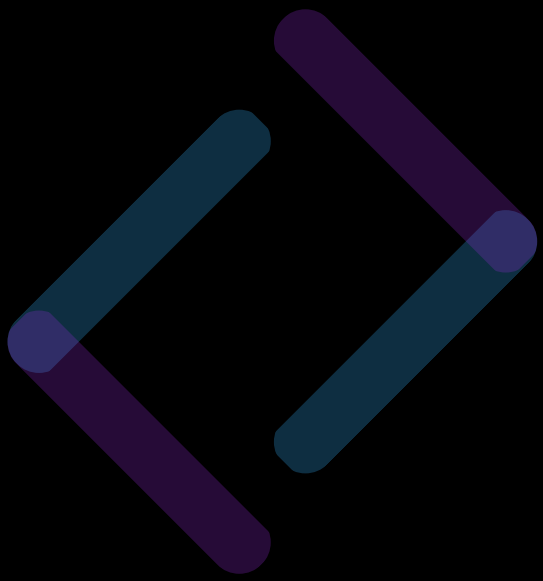
O n1 e n2 equivalem ao x e y da função, respectivamente.

Um segundo exemplo é uma função para achar o quadrado de qualquer número.

main.py

```python
def quadrado(numero):
    return numero * numero
```

Recebendo o número informado e retornando a multiplicação dele por ele mesmo.



PETCode

Escrito e revisado por:
Yan Karlo da Silva Veiga Vasconcellos Dutra
Lara Letittja Sague Lopez Guardiola Velloso
João Gabriel Freitas Acosta

PET Ciências Computacionais