

INTRODUÇÃO A PROGRAMAÇÃO

Software: Conjunto de instruções criado por programadores para definir ações que o computador deva executar; **Algoritmos:** Linhas de códigos que executam instruções para atingir um objetivo.; Algumas linhas serão obrigatórias outras não. Esse é o papel da lógica, o qual consiste em entender quais são ou não são necessários. **Lógica de programação:** são as ordens COERENTE de passos pela qual as instruções são executadas; Vem antes de se escrever o código de programação.

linguagem de programação é uma ferramenta usada para criar um programa, pois é dela que se implementa o algoritmo.

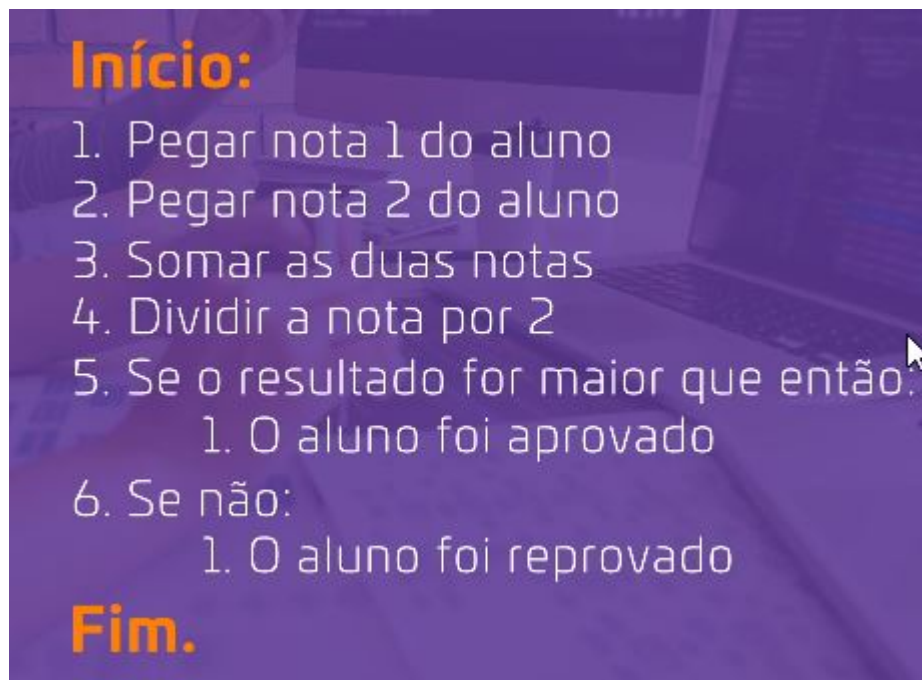
🔑 Ponto chave da lógica de programação é entender quais passos são requisitos para outros, qual melhor caminho e forma para alcançar o objetivo.

TIPOS DE REPRESENTAÇÃO DE UM ALGORÍTMO

Há 2 maneiras básicas, o PSEUDOCÓDIGO e o FLUXOGRAMA

Pseudocódigo: É uma forma genérica de escrever o algoritmo. Ele não leva em conta peculiaridades de linguagens. Objetivo central é entender a lógica sem se prender a questões técnicas.

Ex: Acordar, tomar banho, tomar café outro exemplo:



Fluxograma: forma gráfica de representar um algoritmo. Usando setas e formas geométricas. Facilidade de criação e entendimento. Fluxogramas são compostos de:

- Setas : que representam o caminho do fluxo;
- Retângulos: que são os blocos de códigos;
- Losângulos: que são fluxos que dependem de um valor. Podem ser diferentes em algumas ações.

Exemplo anterior demonstrado em fluxograma



LINGUAGEM E LÓGICA

Cada linguagem tem suas qualidades e seus defeitos. PHP, Java e Node.js : Desenvol Web; Python : ciência de dados; Java, Kotlin e Node.js : Mobile;

Quanto maior a quantidade de pessoas que usam a linguagem mais material terá. Isso facilitará na busca pela resolução de problemas. Uma vez que quanto maior a comunidade maior a interação de experiências.

Python é a linguagem desse curso. Ela tem uma curva de aprendizado curta, fácil, pois é próxima da linguagem humana.

Ver o problema e saber solucionar trará a resposta para saber qual linguagem utilizar.

FORMAS DE PROGRAMAR

Para programar é necessário baixar o compilador. Esse transforma (traduz) a linguagem de programação(alto nível) em linguagem de máquina(baixo nível).

TIPOS DE DADOS CONSTANTES E VARIÁVEIS

Variáveis, é um local na memória que é utilizada para guardar dados, guardam informações importantes, que podem ser acessadas durante a execução de um programa.

Sempre iniciar uma variável com letra ou underline

Há diferentes tipos de variáveis, com locais específicos. Tipos de variáveis.

1. inteiro, int. Número inteiros positivos ou negativos.
2. float, armazenam decimais(.) positivos e negativos.
3. String, caracteres. representado "". linguagem C não tem String.
4. Boolean, valores verdadeiros ou falsos.

o uso de ';' ao final da declaração de variável é opcional no Python. Seu uso se faz quando dois comandos estiverem na mesma linha.

```
5 | print("o valor de b é ", b); print("o valor de c é ", c)
```

No Python não é necessário declarar variável. Ele tem um nível de adaptabilidade maior, reconhece por si só o que é o valor.

Python é CaseSensitive, diferencia maiúsculas de minúsculas

```
numero_inteiro = 5
numero_float = 5.7
texto = "esse é um texto"
booleano = False

print("Este é um tipo")
print(type(numero_inteiro))

print("Este é um tipo")
print(type(numero_float))

print("Este é um tipo")
print(type(texto))

print("Este é um tipo")
print(type(booleano))
```

A função type() em Python informa o tipo da variável que está sendo inserida

```
print("DEMONSTRANDO QUE NÃO HÁ NECESSIDADE DE DECLARAR O TIPO EM PYTHON")
variavel = 5
variavel = 5.7
variavel = "esse é um texto"
variavel = False

print("Este é o 1º tipo")
print(type(variavel))

print("Este é o 2º tipo")
print(type(variavel))

print("Este é o 3º tipo")
print(type(variavel))

print("Este é o 4º tipo")
print(type(variavel))
```

Resultado do código

```
Este é um tipo
<class 'int'>
Este é um tipo
<class 'float'>
Este é um tipo
<class 'str'>
Este é um tipo
<class 'bool'>
DEMONSTRANDO QUE NÃO HÁ NECESSIDADE DE DECLARAR O TIPO EM PYTHON
Este é o 1º tipo
<class 'bool'>
Este é o 2º tipo
<class 'bool'>
Este é o 3º tipo
<class 'bool'>
Este é o 4º tipo
<class 'bool'>
```

LEMBRE-SE Em uma linguagem tipada não é possível mudar o tipo de uma variável após sua declaração. O que não ocorre com o Python que tem tipagem automática. Por exemplo:

```
int v = 5;
v = "teste"
```

isso resultaria em erro, pois numa linguagem tipada não é possível após sua declaração, mudar o tipo da variável. Se fosse em Python, assumiria o valor da última variável.

TIPOS DE OPERADORES

Fazem comparações e operações simples mas fundamentais. São 4 tipos:

1. Atribuição

1. utilizado para armazenar valores nas variáveis. O seu simbolo é o "=".

```
# Exemplo correto de atribuição
a = 5 # variável a recebe valor 5
b = "teste" # variável b recebe texto "teste"
```

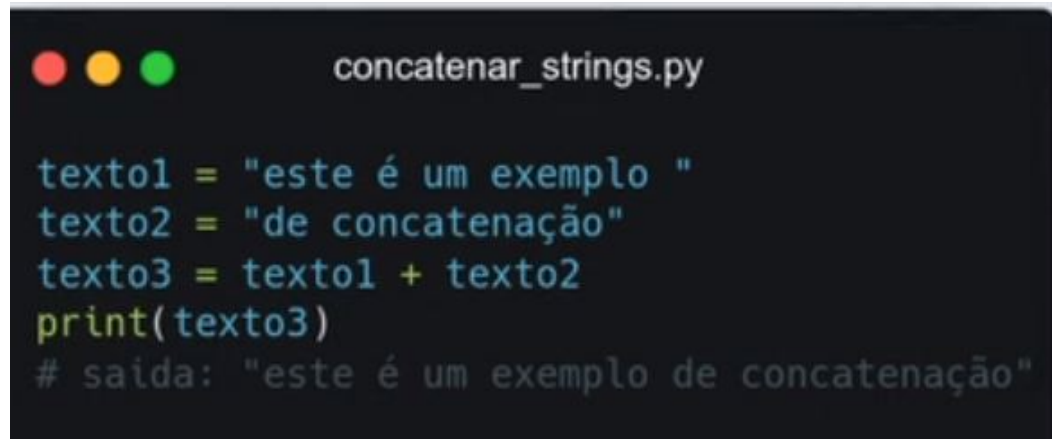
2. Aritmético

Podem ser utilizadas as regras de matemática básica, como o uso dos parênteses.

1. São as operações aritméticas básicas.

1. Soma '+'

1. também pode ser utilizado para concatenar Strings em apenas uma variável.



```
concatenar_strings.py

texto1 = "este é um exemplo "
texto2 = "de concatenação"
texto3 = texto1 + texto2
print(texto3)
# saída: "este é um exemplo de concatenação"
```

2. Subtração '-'
3. Multiplicação '*'
4. Divisão '/'
5. Módulo '%'

1. Que mostra o resto de uma divisão.

3. Relacionais.

Usados para realizar comparações, usam a lógica booleana para determinar se a expressão é verdadeira ou falsa.

1. São os tipos:

1. '<>' Diferente

2. '==' Igual

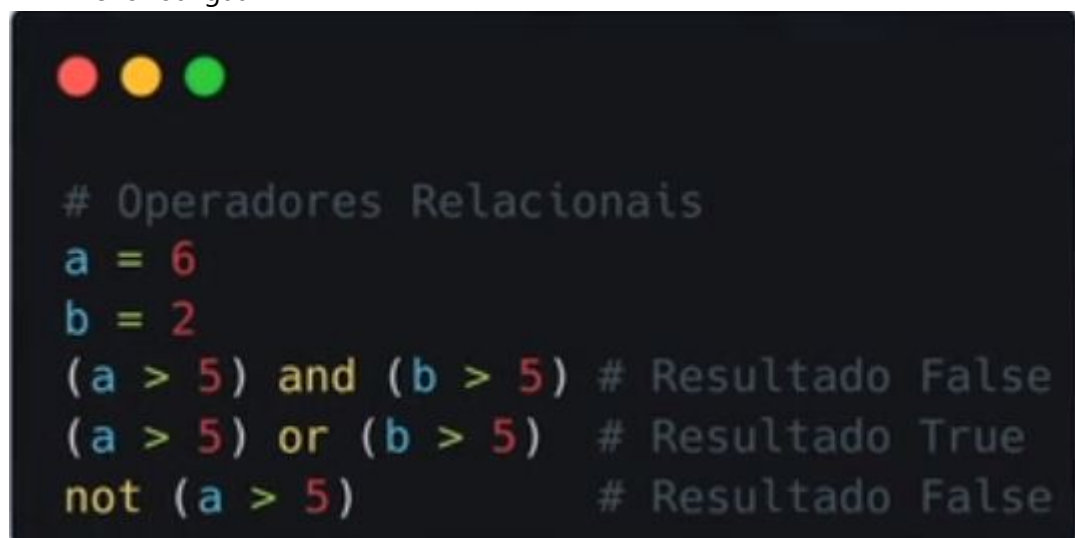
lembre-se: aqui dois sinais, o de atribuição de valor da variável é somente um.

3. '>' Maior que

4. '>=' Maior ou igual

5. '<' Menor que

6. '<=' Menor ou igual



```
# Operadores Relacionais
a = 6
b = 2
(a > 5) and (b > 5) # Resultado False
(a > 5) or (b > 5)  # Resultado True
not (a > 5)          # Resultado False
```

4. Lógicos Utilizados para analisar uma ou mais condições lógicas. O resultado são booleanos. Mais comuns são:

1. E / AND

1. Todas as expressões devem ser verdadeiras para ser verdadeiro

2. OU / OR

1. Só precisa de um verdadeiro para toda a condição ser verdadeira.

3. NÃO / NOT / ! -

1. Inverte a lógica. Se a condição é True, fica False.

ESTRUTURAS DE DECISÃO

O algoritmo segue caminhos distintos a depender da decisão do usuário. Os lógicos e relacionais são muito utilizados na construção de condições.

As estruturas de decisão são :

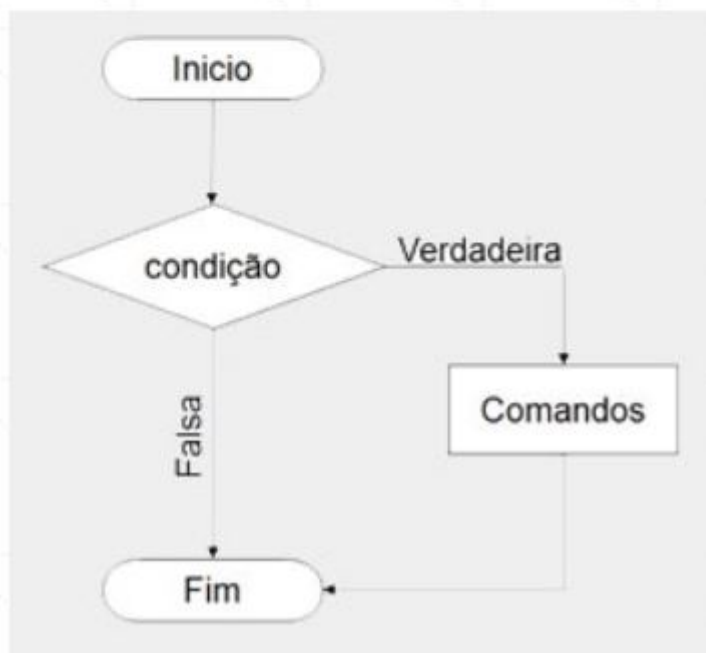
SIMPLES Quando apenas 1 comando é executado e finalizado

Pseudocódigo:

1. INICIO ALGORITMO
2. SE condição ENTAO
3. comando
4. FIM SE
5. FIM ALGORITMO

O comando somente será executado se a condição for verdadeira

Fluxograma:



COMPOSTA a diferença É de que quando a condição não é satisfeita, será executado o outro comando.

Pseudocódigo:

1. INICIO ALGORITMO
2. SE condição ENTAO
3. comando A
4. SENAO
5. comando B
6. FIM SE
7. FIM ALGORITMO

Se a condição for verdadeira, então o comando A será executado. Caso ela seja falsa, apenas o comando B será executado.

Fluxograma:

```
media = (nota1 + nota2) / 2
```

```
se (media >= 7) então
```

```
  a. Aprovado
```

```
se (media < 7) então
```

```
  a. Reprovado
```

SENÃO

No exemplo anterior, foram aplicadas duas condições e só uma será executada, são elas: a média maior ou igual a sete; e a média menor que sete.

Ou seja, se a média do aluno for maior ou igual a sete, ele está aprovado, senão, está reprovado. O termo senão é utilizado na programação para facilitar a compreensão dos comandos.

```
media = (nota1 + nota2) / 2

se (media >= 7) então

a. Aprovado
Senão

a. Reprovado
```

SENÃO SE

E se a ação possuir **mais de duas possibilidades**? Nem sempre será possível simplesmente aplicar vários "se", pois isso pode acarretar alguns problemas.

O próximo exemplo apresenta um pseudocódigo que recebe a média do aluno como entrada e armazena a categoria de nota como letras.

```
media = 8

se (media == 10) então
a. mediaLetra = "A"

se (media >= 9) então
a. mediaLetra = "B"

se (media >= 8) então
a. mediaLetra = "C"

se (media >= 7) então
a. mediaLetra = "D"

Senão
a. mediaLetra = "F"
```

De início, **"C"** aparenta ser o valor armazenado na variável mediaLetra, mas, na quinta linha, a condição também é verdadeira, pois o valor da variável media é maior que sete. Então, **o valor armazenado na variável seria "D"**.

Isso acontece pois **todos os "se" são independentes**, sendo necessário inseri-los em uma condição **para que apenas um deles seja executado**. Para isso, é utilizado o **"senão se"**.

Como mostra o próximo exemplo, quando o "senão se" é utilizado no lugar de cada "se", se mantém o primeiro "se" e **quando uma condição "senão se" for a correta, as outras não serão mais testadas**. Observe o exemplo a seguir.


```
media = 8
se (media == 10) então
a. mediaLetra = "A"
senão se (media >= 9) então
a. mediaLetra = "B"
senão se (media >= 8) então
    a. mediaLetra = "C"

senão se (media >= 7) então
a. mediaLetra = "D"
senão
a. mediaLetra = "F"
```

CONDIÇÕES MAIORES

Pode-se utilizar **operadores lógicos** nas estruturas condicionais. No exemplo anterior, pode-se fazer uso desses operadores dentro dos "se".

```
media = 8

se (media == 10) então
a. mediaLetra = "A"

senão se (media >= 9) E (media < 10) então
a. mediaLetra = "B"

senão se (media >= 8) E (media < 9) então
a. mediaLetra = "C"

senão se (media >= 7) E (media < 8) então
a. mediaLetra = "D"

senão
a. mediaLetra = "F"
```

Assim, **o problema comentado anteriormente não ocorreria**, mesmo se não fossem utilizados os "senão se".

CONDIÇÕES EM PYTHON

Se CONDIÇÃO **então** -> **if** CONDIÇÃO :

Os ':' significam **então**

Senão se CONDIÇÃO **então** -> **elif** CONDIÇÃO :

Todos os comandos dentro do "if" precisam ser indentados, pois a **Python é uma linguagem indentada**.

IDENTAÇÃO NO PYTHON

Obrigatória em Python. Torna o código mais legível, assim como é responsável em manter a boa execução das instruções. Caso não respeite pode até gerar erro.

Por exemplo, no caso de um `if`, o que determina se um código está dentro da condicional é o fato dele ter sido indentado. O **Código 7** apresenta um exemplo onde o comando `print` só será executado se o valor da variável `x` for maior que 8.

```
1 | x = 10
2 |
3 | if x > 8:
4 |     print("x é maior que 8")
```

Código 7. Comando print dentro de um if

Agora veja esse novo exemplo no qual o comando `print` será executado de qualquer forma por não estar indentado e, por isso, não estar contido no `if`.

```
1 | x = 10
2 |
3 | if x > 8:
4 |
5 | print("x é maior que 8")
```

Código 8. Comando print fora de um if

USO DE PARENTESSES EM PYTHON NOS COMENDOS IF ELSE WHILE

São opcionais.

```
1 | x = 10
2 | if (x > 8):
3 |     print("x é maior que 8")
```

Código 9. Estrutura de controle de fluxo com parênteses

```
1 | x = 10
2 | if x > 8: # nessa linha removemos os parêntesis
3 |     print("x é maior que 8")
```

Código 10. Estrutura de controle de fluxo sem parênteses

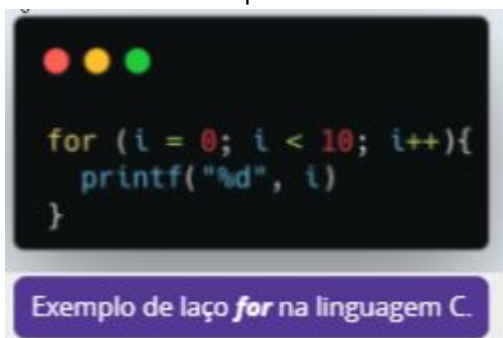
APRENDER A EXECUTAR LAÇOS DE REPEDIÇÃO

Na programação existem 2 TIPOS: FOR e WHILE **FOR** que é um **laço determinado** com um número certo de repetições. **WHILE** que é um **laço inderteminado** <https://vaiprogramar.com/como-usar-os-lacos-de-repeticao-for-e-while-em-python/>

FOR

O for em Python é diferente, não precisa ser iterado sobre números inteiros como o java. Em Python, o laço for é executado **sobre uma coleção de objetos**

mais utilizado na programação. **DEPENDE de uma variável contador**. Elementos necessários são: 1. Definir a faviável contador. 2. Definir o valor de parada. 3. Definir se o contador receberá um incremento ++ ou



desencremento --.

No Python é mais simples. Se usa o **range()** que



já se autoincrementa com o contador.

for VARIÁVEL in range(LIMITE):

Sintaxe:

Função range()

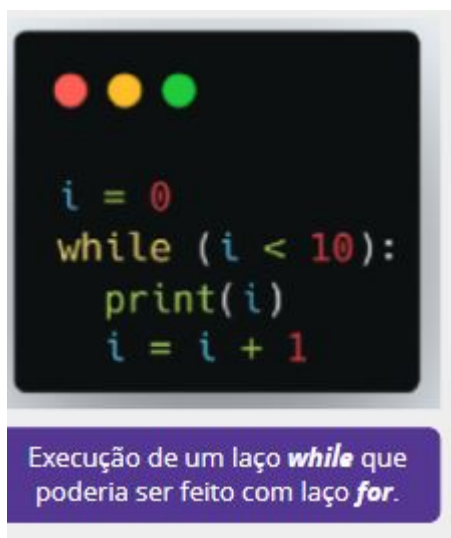
Por padrão, a função range() **inicia em 0 e vai incrementando em 1 até o valor determinado**, que é o critério de parada. Também, é possível adicionar outras opções, parâmetros, na função range().

- **Iniciar por outro valor que não seja 0:**
`range(2, 10)` começa no número 2 e para quando chegar em 10, incremento em 1;
- **Mudar o incrementador:** `range(0, 10, 2)` começa no número 0 e para quando chega em 10, porém, com o incrementador de 2 em 2;
- **Redução e não incremento:** `range(10, 0, -1)` começa a partir do 10 e para quando chega em 0, reduzindo de 1 em 1.

WHILE

O laço while **é mais genérico**, pois ele **permite outros tipos de repetição** quando necessário. Então, *os comandos que estiverem no bloco while serão executados enquanto a condição for verdadeira*. Sua estrutura na Python é while condição:.

Também é possível **utilizar o while da mesma forma que o for**, porém, é necessário incrementar ou diminuir o contador manualmente, tal qual como o exemplo da contagem feita com while. Como na maioria das vezes são repetições que usam um contador, o for é mais utilizado que o while.



do while (NÃO VI EM PYTHON)

É semelhante ao while comum, mas ele garante que o código seja executado no mínimo uma vez, fazendo a



checagem no final da execução.

break e continue

break: *interrompe todo o laço que está em execução.* Geralmente, é usado junto com um if adicional para condições específicas;

continue: *finaliza a execução atual do laço e move para a próxima.* Geralmente, é usado quando não se deseja mais executar uma parte do código.



FUNÇÕES

Função dentro da programação lembra muito o refrão das músicas. Uma parte a ser repetida, porém só é colocada na música o termo "Refrão". Funções são blocos de códigos que executarão uma parte específica do código. Razões para seu uso: **evitar repetição** - as funções são utilizadas em mais de um lugar do código, então elas evitam a repetição e facilitam o entendimento do código;

encapsulamento - cada função tem uma tarefa específica e, geralmente, seu nome remete a isso. Assim, quando alguém ler o código, entenderá o que e como será feito.

ESTRUTURA DA FUNÇÃO

A estrutura de uma função é composta por **três elementos**.

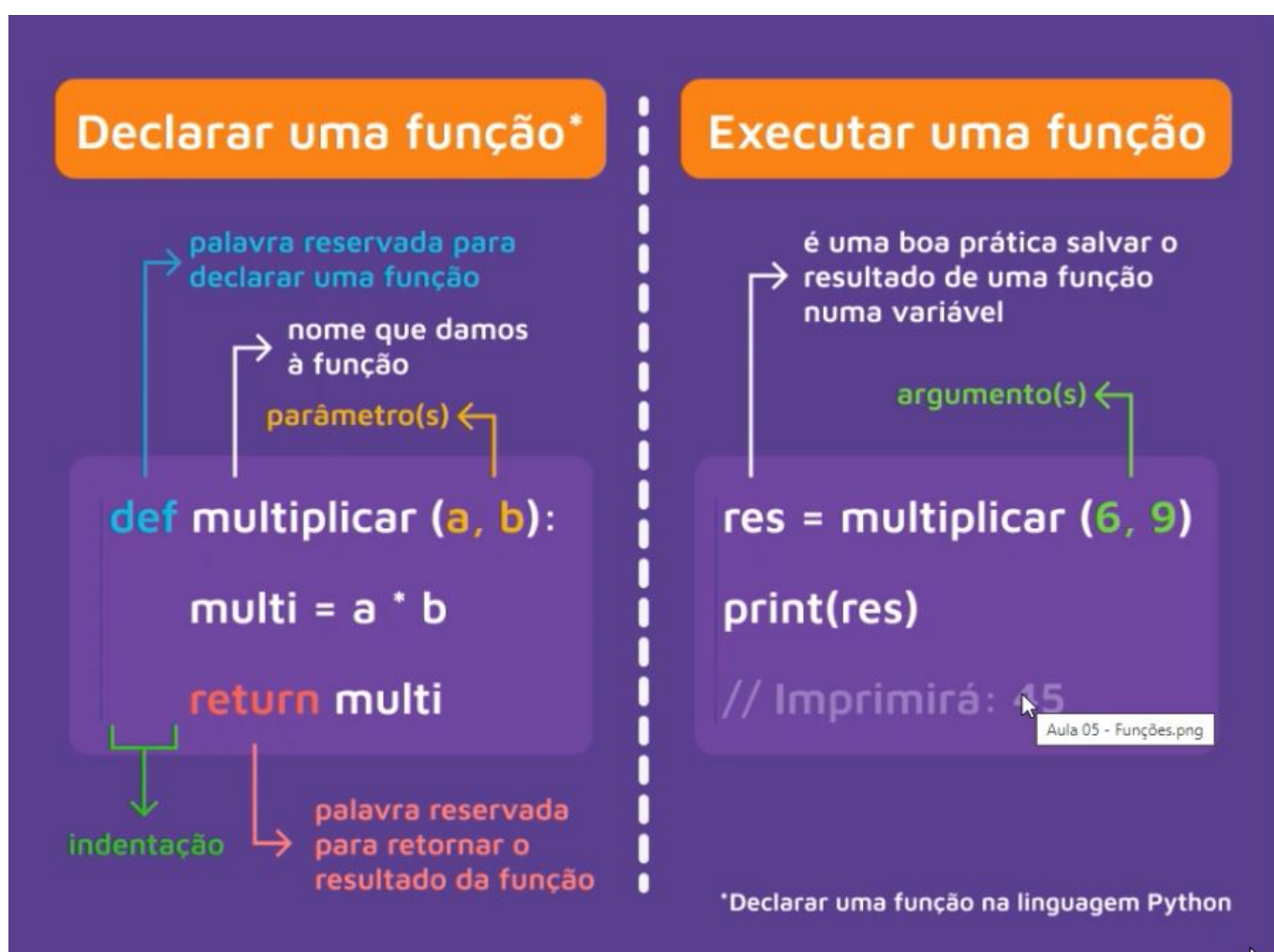
Nome da função: é interessante ter um nome que defina a tarefa da função. Por ele que a função é chamada no meio do código;

Parâmetros: é necessário informar quais valores serão utilizados na função, pois eles podem ser usados de acordo com a situação, ficando diferentes;

Resposta (retorno): a função deve retornar o resultado da sua operação ao terminar a execução.



Na linguagem Python, a estrutura da assinatura da função é "**def NOME_FUNCAO (PARAMETROS):**" e vale lembrar da indentação da Python.





return determina o valor que será retornada de toda a função.

VARIÁVEIS LOCAIS E GLOBAIS

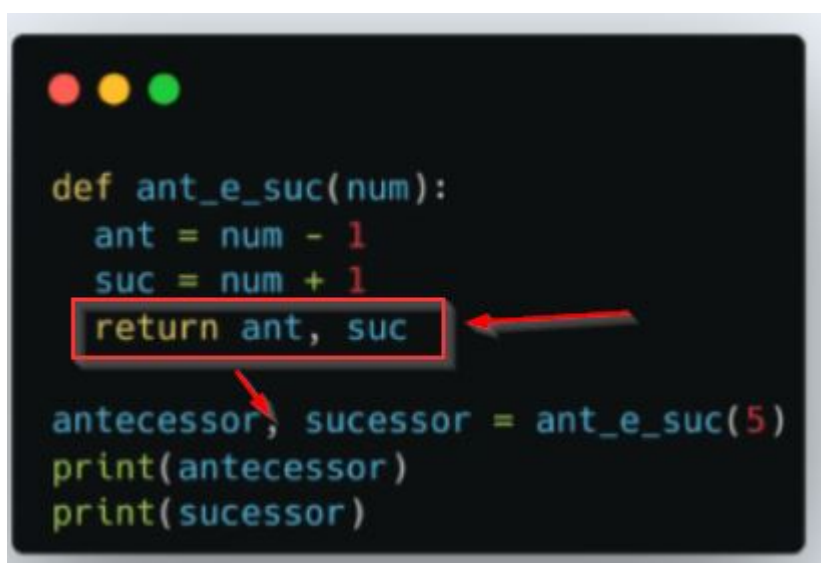
As variáveis que podem ser acessadas por todo o código são chamadas de variáveis globais, mas seu uso excessivo não é uma boa prática. Ao enviar um valor como parâmetro na função, ele se torna uma variável local e passa a ser acessada somente dentro da função, não prejudicando o funcionamento do resto do sistema.

IMPRIMIR DENTRO DE UMA FUNÇÃO

Não é uma boa prática. uma vez que o objetivo das funções é, apenas, executar um procedimento e retornar o resultado. **mas é recomendado exercê-la**, pois facilita o desenvolvimento.

RETORNO DA FUNÇÃO

Na linguagem **Python**, é possível retornar mais de um valor na função, basta separar os valores por vírgula quando for receber o valor do retorno,



No exemplo, há um código simples, que pega o valor passado no parâmetro e retorna seus números antecessor e sucessor. Perceba o uso da vírgula no retorno da função e no resultado do código principal.

FUNÇÕES PYTHON

Exercício: Faça uma função calculadora de dois números com três parâmetros: os dois primeiros serão os números da operação e o terceiro será a entrada que definirá a operação a ser executada. Considere a seguinte definição:

1. Soma
2. Subtração
3. Multiplicação
4. Divisão

Caso seja inserido um número de operação que não exista, o resultado deverá ser 0. Resolução:

```
def calculadora(num1, num2, operacao):  
    if (operacao == 1):  
        return num1 + num2  
    elif (operacao == 2):  
        return num1 - num2  
    elif (operacao == 3):  
        return num1 * num2  
    elif (operacao == 4):  
        return num1 / num2  
    else:  
        return 0  
  
resultado = calculadora(4, 7, 1)  
print(resultado)
```

Neste exemplo vale lembrar que definimos o nome dos parâmetros como 'num1', 'num2' e 'operacao', porém, é possível dar qualquer nome à eles, respeitando as boas práticas de evitar caracteres especiais e letras maiúsculas. No bloco de código da função usamos estruturas condicionais que verificam o valor (código) da operação para retornar uma operação matemática distinta. Além disso, incluímos um caso 'else' que retorna 0 caso o código da operação fornecido não foi previamente definido. Finalmente, executamos a função e salvamos o resultado dela numa variável, para depois imprimi-la com a função 'print()'. Você pode testar a função trocando os argumentos passados na execução da função calculadora.

FLUXO DE ENTRADA E SAÍDA DO USUÁRIO

No mundo real, os sistemas foram feitos para terem algum tipo de interação humana ou de outro sistema.

Na programação há uma área focada na construção de telas e experiência do usuário, que tem por objetivo auxiliar o usuário, essa área se chama **Front-End**

Utilizaremos para entrar o dado o termo **input()**, o programa pausa sua execução aguardando o dado a ser inserido.

Input com laços de repetição

O laço while vai ficar sendo executado até que a variável tecla possua a letra "f"

Laço que execute todo o seu código até que o usuário realize um determinado comando

```
tecla = ""
while tecla != "f":
    print("Digite qualquer tecla, exceto o f")
    tecla = input()
print("Fim do laço")
```

Enquanto o usuário apertar qualquer outra tecla o código continuará rodando

Por mais que a Python faça a análise do tipo de variável automática, no caso do input, **toda entrada será String**.

Se tentar somar as entradas, sem realizar uma conversão dará erro.

```
print("Digite um número")
numero = int(input())
res = numero + 5
print("O número digitado mais 5 é igual a " + str(res))
```

converteu o input() em inteiro

retornou o resultado para str para concatenar

Exercício Faça uma função calculadora que os números e as operações serão feitas pelo usuário. O código deve ficar rodando infinitamente até que o usuário escolha a opção de sair. No início, o programa mostrará a seguinte lista de operações:

1: Soma

2: Subtração

3: Multiplicação

4: Divisão

0: Sair

Digite o número para a operação correspondente e caso o usuário introduza qualquer outro, o sistema deve mostrar a mensagem "Essa opção não existe" e voltar ao menu de opções.

Após a seleção, o sistema deve pedir para o usuário inserir o primeiro e segundo valor, um de cada. Depois precisa executar a operação e mostrar o resultado na tela. Quando o usuário escolher a opção "Sair", o sistema irá parar.

É necessário que o sistema mostre as opções sempre que finalizar uma operação e mostrar o resultado.

```
def calculadora(num1, num2, operacao):  
    if (operacao == 1):  
        return num1 + num2  
    elif (operacao == 2):  
        return num1 - num2  
    elif (operacao == 3):  
        return num1 * num2  
    elif (operacao == 4):  
        return num1 / num2  
    else:  
        return 0  
  
executar = True  
  
while (executar == True):  
    print("Qual operação você quer realizar?")  
    print("1: Soma 2: Subtração 3: Multiplicação 4: Divisão 0: Sair")  
    operacao = int(input())  
    if(operacao < 0) or (operacao > 4):  
        print("Essa opção não existe")  
    elif(operacao == 0):  
        executar = False  
    else:  
        print("Insira o primeiro número da operação:")  
        num1 = int(input())  
        print("Insira o segundo número da operação:")  
        num2 = int(input())  
        resultado = calculadora(num1, num2, operacao)  
        print("O resultado é:", resultado)
```

Para esta atividade aproveitamos a função 'calculadora' criada no CodePark anterior. Além de definir ela, criamos uma variável 'executar' que servirá para dizer ao nosso programa quando deve parar de executar. Definimos ela com o valor inicial 'True' e definimos uma estrutura de repetição 'while' que continuará executando o bloco de código dela sempre e quando a variável 'executar' guardar o valor 'True', por isso é importante que o valor inicial dela seja 'True', senão nosso sistema não vai iniciar.

No bloco de código da estrutura de repetição imprimimos mensagens orientando o usuário com o comando de saída 'print()', e capturamos os valores inseridos pelo usuário com o comando de entrada 'input()'. Vale

lembrar que os dados inseridos pelo usuário com o comando 'input()' por padrão serão do tipo string

Como faremos operações matemáticas com esses dados, precisamos "envolver o comando input" pela função 'int()' (em outras palavras, passar o retorno da operação 'input()' como argumento da função 'int()'). Assim a função 'int()' receberá os dados inseridos em formato de string, e os retornará em formato de número inteiro para serem guardados nas variáveis 'num1' e 'num2'.

Na sequência, verificamos se a opção escolhida pelo usuário é menor do que 0 ou maior do que 4 pois, caso o usuário escolher qualquer opção que cumpra essas condições, o restante do bloco de código não deve ser executado, senão que devemos retornar ao começo da nossa estrutura de repetição.

Depois definimos um 'elif()' onde verificamos se a opção escolhida é 0 e mudamos o valor da variável 'executar' para 'False' pois assim a condição de parada da estrutura 'while' deixa de ser verdadeira e nosso programa irá parar. Finalmente, definimos um 'else' onde capturamos os dados inseridos pelo usuário, convertemos eles para números inteiros, executamos nossa função 'calculadora()', e imprimimos o resultado que ela retorna.

TRATAMENTO DE ERROS

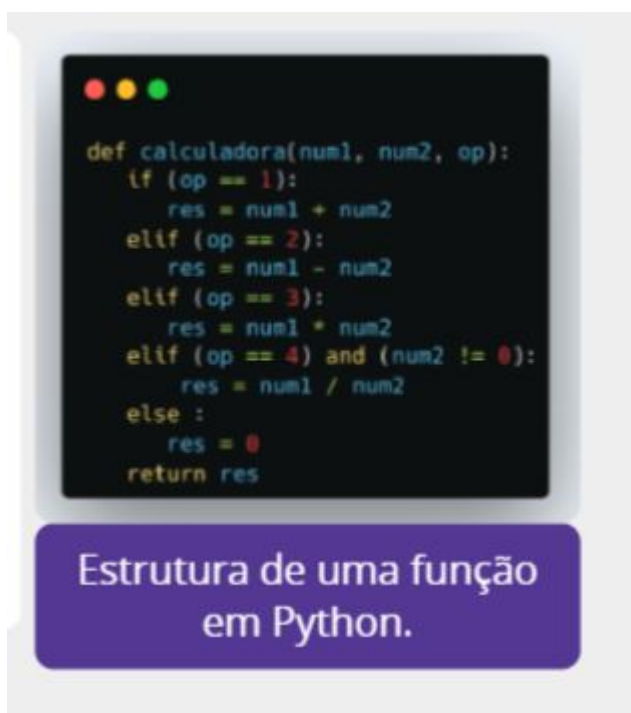
Usuários informam muitas vezes valores não suportados pelo sistema. Ocasiona erro.

Exemplo: uma letra no lugar de um número

USAR IF PARA RESOLVER PROBLEMAS DE ERRO

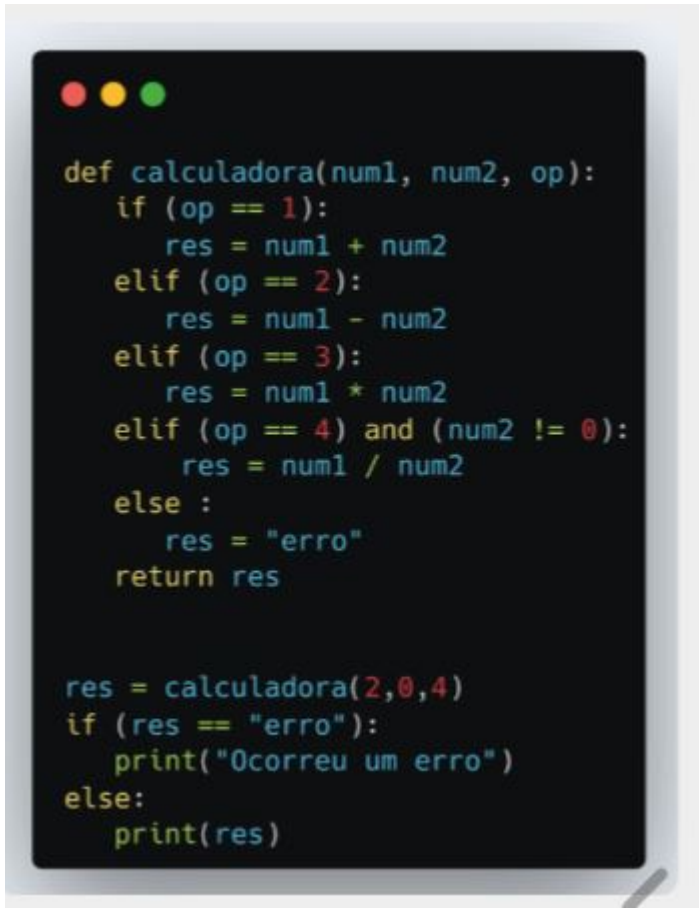
Um if, por exemplo, pode prevenir muitos problemas.

No caso da calculadora, a adição de mais uma condição na operação de divisão para checar se o segundo número é diferente de zero, resolve o problema.



Percebe-se que o código já possui um tratamento de erro, pois ele retornaria o valor "0" caso nenhuma das opções possíveis fossem preenchidas. Porém, "0" também pode ser um resultado válido.

Então, **lembrando que a Python tem tipagem dinâmica**, é possível retornar uma mensagem de erro ao invés de um número.



```
def calculadora(num1, num2, op):  
    if (op == 1):  
        res = num1 + num2  
    elif (op == 2):  
        res = num1 - num2  
    elif (op == 3):  
        res = num1 * num2  
    elif (op == 4) and (num2 != 0):  
        res = num1 / num2  
    else :  
        res = "erro"  
    return res  
  
res = calculadora(2,0,4)  
if (res == "erro"):  
    print("Ocorreu um erro")  
else:  
    print(res)
```

Acima, no novo exemplo, há uma condição no código principal que checa se o resultado da função é um erro ou não. Portanto, é possível adaptar cada vez mais esse erro. É possível ter uma mensagem de erro caso seja uma operação indevida ou uma divisão por zero.

NEM TODO ERRO RESOLVE COM IF - EXCEPTIONS

O uso da condição if auxilia no tratamento de erros, mas, em alguns casos, não é possível utilizá-lo ou seu uso tornaria o código mais complicado de ser desenvolvido. Por exemplo, ao tentar converter o input do usuário para um número inteiro. Não há como checar antes se o valor é um número de fato ou se é um texto, só depois que o erro acontece, então, nesse caso, é impossível utilizar um if para resolver o problema.

EXCEPTIONS

As estruturas de exceção, exceptions, são utilizadas para tratar erros inesperados que podem aparecer durante o código. Elas possuem a seguinte estrutura:

```
try: CODIGO  
except CODIGO
```

Lembrete: **Ao ocorrer o erro, o programa NÃO PARA DE EXECUTAR**

Essas estruturas preparam o código para executarem ações caso erros predeterminados ocorram. Assim, **evitam que o erro interrompa a execução do programa**

```
print("Insira um numero")
try:
    valor = int(input())
    valor = valor + 5
    print("O valor digitado + 5 e igual a " + str(valor))
except:
    print("nao foi digitado um numero")
```

No exemplo acima, o código solicita que seja inserido um número e, caso não seja colocado um número inteiro, ele apenas retorna que não foi digitado um número e nenhum erro é mostrado.

Todo o código presente dentro do try é executado e, se algum erro dentro dele acontecer, o código, automaticamente, executará o que estiver dentro do except.

Resumidamente, o código tentará fazer tudo, exceto se algo ocorrer de errado.

Documentação Python para tratamento de exceções <https://docs.python.org/pt-br/3/tutorial/errors.html>

PRATICANDO EXCEPTIONS

Problema: Desenvolva um programa que só deve aceitar números pares. Siga as seguintes instruções:

caso um número ímpar seja digitado, informe ao usuário que ele digitou um valor ímpar e peça novamente por um número par;

caso uma letra seja digitada, informe ao usuário que ele digitou um caractere inválido e peça novamente por um número par.

Solução:

```
numeroCorreto = False

while (numeroCorreto == False):
    print("Insira um número par")

    try:
        numero = int(input())

        if (numero%2 == 0):
            numeroCorreto = True

            print("Você digitou um numero par !")

    else :
```

```
        print("Você digitou um número impar")

    except:

        print("Caracter inválido, por favor digite um número par")
```

Relembre as instruções do projeto

Desenvolva um programa que recebe do usuário nome completo e ano de nascimento que seja entre 1922 e 2021.

A partir dessas informações, o sistema mostrará o nome do usuário e a idade que completou, ou completará, no ano atual (2022).

Caso o usuário não digite um número ou apareça um inválido no campo do ano, o sistema informará o erro e continuará perguntando até que um valor correto seja preenchido.

Gabarito

Exemplo de código:

```
print("Digite seu nome:")
nome = input()

executar = True

while(executar == True):
    print("Digite seu ano de nascimento")
    try:
        ano = int(input())
        if(ano < 1922) or (ano > 2021):
            print("O ano precisa ser entre 1922 e 2021")
        else:
            idade = 2022 - ano
            print("O usuário", nome, "completou ou completará", idade, "anos de idade em 2022")
            executar = False
    except:
        print("Os anos precisam ser escritos apenas com números")
```

Como nosso sistema não faz nenhum tipo de validação para o nome do usuário, podemos começar o código do nosso sistema solicitando o nome do usuário e guardando ele na variável 'nome'. Depois disso declaramos

a variável 'executar' com o valor inicial 'True' para controlar quando nosso sistema deverá parar.

Na sequência, definimos uma estrutura de repetição para que nosso sistema continue solicitando o ano de nascimento até o usuário informar um valor válido. No bloco de código da estrutura 'while' começamos solicitando o ano de nascimento do usuário e, na sequência, definimos um bloco 'try... except' para evitar que nosso programa pare ao encontrar erros inesperados, como uma letra ao invés de um número.

Podemos definir no 'except' a resposta ao usuário solicitando que o ano seja escrito apenas com caracteres numéricos. Já dentro do 'try' solicitamos o ano de nascimento do usuário e usamos uma estrutura condicional onde imprimimos uma mensagem solicitando que insira um ano válido caso o mesmo não esteja dentro do intervalo 1921-2021.

Finalmente, se o usuário informar um ano válido, executamos o 'else', onde é calculada a idade do usuário e imprimimos uma frase concatenando o seu nome e sua idade, e mudamos o valor da variável 'executar' para 'False' para parar o sistema.

Try Bloco de código Except: print(erro generico)

Try

REVISÃO: INTRODUÇÃO À PROGRAMAÇÃO

Pseudocódigo:

- Os passos podem ser representados em uma sequência numérica
- Não leva em consideração as particularidades de cada linguagem de programação

Fluxograma:

- Utiliza setas para representar o fluxo de informação
- Utiliza retângulos e losangos para representar blocos de código e estruturas condicionais respectivamente

Ferramentas de desenvolvimento:

- Linguagem de programação
- Compilador
- Editor de texto ou IDE

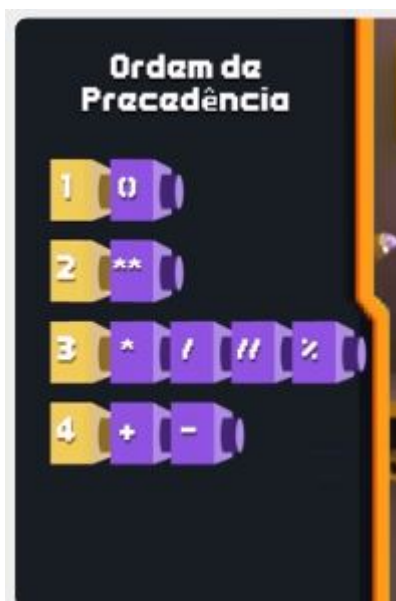
<https://view.genial.ly/635878801ea7ae001956d98b>

Operadores



Ordem de precedência: Quem vem primeiro na execução numa expressão.

1. '()' parênteses;
2. '**' potência;
3. '*' multiplicar; '/' dividir; '//' divisão inteiro; '%' divisão módulo(resto); Se tiver eles ao mesmo tempo numa expressão, resolve a que aparecer primeiro.
4. '+' soma; - subtração;



Calcular raiz quadrada de um número é o mesmo que elevar um número a potência 1/2. Ex: $25^{1/2}$; raiz cúbica $127^{1/3}$

No Python, pode-se fazer `"*5`, que ele vai repetir o igual 5 vezes;

Várias dicas no Python

The image shows a Python IDE with a file named `aula06a.py`. The code defines two variables, `n1` and `n2`, and performs arithmetic operations. It uses `print` statements with `format` and `end` parameters. Annotations explain the use of `\n` for line breaks, `end=""` to prevent automatic line wrapping, and `:.3f` for float formatting. The output shows the results of these operations, with the first `print` statement spanning multiple lines due to the `\n` and `end=""` usage.

```
1 n1 = int(input('Um valor: '))
2 n2 = int(input('Outro valor: '))
3 s = n1 + n2
4 m = n1 * n2
5 d = n1 / n2
6 di = n1 // n2
7 e = n1 ** n2
8 print('A soma é {}, \n o produto é {} e a \n divisão é {:.3f}'.format(s, m, d), end='')
9 print('Divisão inteira {} e potência {}'.format(di, e))
```

Annotations:

- quebra a linha print (points to `\n` in line 8)
- delimita em 3 casas o resultado em float (points to `:.3f` in line 8)
- uni em só uma linha dois print() No campo em branco pode ser inserido qualquer caractere para fazer parte da união. No ex.: união s/ caractere (points to `end=""` in line 8)
- {}chaves, informa que terá um valor indicado pelo .format. No caso recebeu 2 valores de variáveis (points to `{}` in line 8)
- estariam numa linha se não fosse o \n (points to `\n` in line 8)

Output:

```
/Library/Frameworks/Python.framework/Versions/3.6/bin/python3.6 /Users/.../aula06a.py
Um valor: 4
Outro valor: 8
A soma é 12,
o produto é 32 e a
divisão é 0.500 Divisão inteira 0 e potência 65536
Process finished with exit code 0
```

Todas as estruturas de repetição tem 3 elementos em comum:

1. variável contador
2. condição de parada
3. incrementador