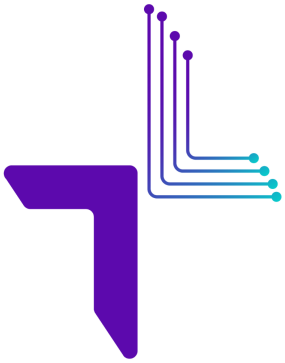
APOSTILA I

PROGRAMAÇÃO ALÉM DO CÓDIGO

REALIZAÇÃO:



Conteúdo

1. Introdução, Tipagem e Funções de E/S

1.1 O que é programação?

Para iniciar de fato o estudo de programação é necessário entender o que de fato é programação. A programação pode ser definido como conjunto de comandos dado por um usuário (programador, ou como você quiser chamar) para cumprir uma certa ação. Quando pensamos em programação pensamos primeiramente naqueles códigos extensos em um computador ou uma sequência infinita de zeros e uns verdes em uma tela verde (estilo matrix), mas podemos pensar a programação como sendo qualquer ordem dada para se chegar a certo objetivo, ou seja, você programou a vida toda sem ao mesmo saber o que era programação, vamos exemplificar: você acordou de manha, dormiu demais e percebe que só faltam 15 minutos para dar a hora do início do seu expediente, bem você tem um objetivo, que é chegar pronto no seu trabalho em 15 minutos, mas para isso você precisa realizar uma série de ações: escovar os dentes, tomar banho, tomar café da manhã, se deslocar até o local de trabalho, isso tudo com o detalhe que tem que ser rápido já que faltam 15 minutos. Pois bem, todos esses comandos que eu dei são de certas forma um modelo de programação, já que eu tenho várias tarefas que vão ser executadas de certa maneira para eu cumprir um certo objetivo, isso é chamado de **algoritmo**.

1.2 Algoritmo ou programa?

Se você observar o programa nada mais é que um algoritmo. Para desenvolvermos os programas nós utilizamos as chamadas **linguagens de programação**.. A linguagem que estamos aprendendo agora é Python, que nada mais é que uma ferramenta utilizada para converter um algoritmo que escrevemos na linguagem natural para um algoritmo em uma linguagem mais próxima daquela que o computador entende (linguagem de máquina). Assim, um programa nada mais é do que um algoritmo convertido da linguagem natural para as instruções em uma linguagem compreendida pelo computador, utilizando-se para

isso das linguagens de programação.

1.3 Tipos de dados

Um tipo de dado especifica um conjunto de valores, determinando sua natureza, seu tamanho, sua representação e sua imagem. Algumas características como as que seguem servem para descrever um tipo. É preciso destacar que o tamanho de um tipo de dado não é pré-determinado, pois existe muitos fatores a se considerar, como a arquitetura do computador e o compilador que está sendo usado. Natureza: Caracteriza o tipo representado, que pode ser, por exemplo, um caractere, um número inteiro ou real. Tamanho: Determina o tamanho em bits necessários para armazenar os valores do tipo. Representação: Determina a forma como os bits armazenados devem ser interpretados.

1.4 Tipos de variáveis

- **Int:** guarda números inteiros;
- **Float:** guarda números racionais (ou "números quebrados");
- **String:** Guarda letras e/ou palavras;
- **Boolean:** Guarda funções lógicas de verdadeiro ou falso.

Vale ressaltar que você pode declarar a qualquer momento no seu programa uma variável fazendo ela valer algum valor ou não, a forma de usar e declarar se assemelha bastante de como se faz na matemática.

1.5 Funções de entrada e saída

A entrada de dados nada mais é que a maneira com que o dado é inserido no programa que você está fazendo, já a saída como o próprio nome diz é para onde os dados vão após o processamento. O editor de texto, por exemplo o bloco de notas, está recebendo de entrada cada letrinha que se tecla no teclado e processando-a para formar um texto, exibido na tela do monitor. Dessa forma, nós podemos observar que o teclado funciona como um dispositivo de entrada enquanto o monitor funciona como o dispositivo de saída. Esse é o cenário mais comum para um programador, porém temos outros exemplos no dia-a-dia, você sabe quais?

Como em uma ligação, que o microfone é o dispositivo de entrada que processa a voz e a emite no alto-falante, que é o dispositivo de saída, ou em um carro que o sensor de velocidade é dispositivo de entrada para o programa responsável pelos sensores de velocidade e os exibe no painel do veículo, sendo, portanto, o dispositivo de saída.

Agora vem a pergunta: Como que eu faço funções de entrada e saída enquanto eu estiver programando? A resposta vai depender da linguagem que você estiver usando, como nesse curso estamos usando o Python como linguagem base, a função de entrada do Python vai ser:

```
print('digite o que quiser')
```

Em que o que fica dentro do parênteses e as aspas vai ser o que vai ser escrito na tela. Isso também serve para caso você quiser mostrar uma variável, nesse caso você teria que colocar a variável dentro do parênteses sem as aspas. Exemplo:

```
A = 2
print(A)
```

Nesse caso acima o programa iria mostrar o valor da variável, que no caso é 2. Você pode fazer também com que o usuário que abrir o programa digite um número para ser armazenado em uma variável, em vez de você colocar, a estrutura para fazer isso é a seguinte:

```
A = input('digite um numero:')
print(A)
```

No caso acima o usuário que abriu o programa vai digitar um valor qualquer que vai ser armazenado na variável A e depois seria mostrado na tela.

1.6 Problemas

1) Escreva "Olá, mundo!"na tela.

2) Faça uma calculadora que realize apenas a operação de soma entre dois números quaisquer.

3) Faça um programa que leia a idade do usuário e imprima da seguinte forma: “A sua idade é: ".

4) Faça um programa que leia o dia, o mês e o ano inserido por um usuário e imprima esses dados no formato convencional brasileiro (“dia/mês/ano”) e no norte-americano de data (“mês/dia/ano”).

5) Faça um programa que leia 4 notas de um usuário e mostre a média aritmética dessas notas.

2. Condicionais e loop

Aqui iremos aprender como fazer o nosso programa tomar decisões a partir de determinadas condições. Utilizando if, else e , é possível definir se o algoritmo seguirá um caminho ou outro. Imagine que você está dirigindo um carro num dia de chuva e chega em um ponto que se depara com uma bifurcação (opção de dois caminhos para seguir), nesse momento você faz a pergunta para si mesmo: A rua à direita está alagada? Se a resposta for não, você segue nela, do contrário, faz-se a mesma pergunta para a rua da esquerda, se a resposta for não, você segue nela, se for sim, você faz o retorno e busca outro caminho. Analogamente, os comandos de tomadas de decisão funcionam da mesma maneira, programamos para que em um ponto do código o algorítmo "se questione" sobre algumas condições pré estabelecidas para tomar decisões baseadas nas respostas.

2.1 IF

O comando IF é utilizado para determinarmos um "se". Se a condição nos parênteses for verdadeira, o programa executa o que estiver dentro das chaves, caso contrário, ele segue. Será nesses parênteses que comumente utilizaremos os operadores relacionais.

2.2 Else

O comando else é dependente do IF, ou seja, caso o if retorne falso, será executado o que estiver dentro das chaves do else, logo, podemos concluir que para todo else sempre terá um if, mas nem todo if precisará ter um else.

2.3 O que é um loop?

Um loop nada mais é do que uma estrutura que permite a repetição de um determinado bloco de comandos várias vezes. Explicando de forma mais precisa, em um loop você

poderá executar uma serie de comandos enquanto uma condição não for quebrada. Os loops também são chamados de laços ou laços de repetição ou estruturas de repetição.

2.4 While

É o tipo de loop mais simples de todos. While significa "enquanto", e como o próprio nome diz, esse loop segue estritamente o modelo de repetir um bloco de comandos enquanto uma condição for verdadeira.

Nessa estrutura é necessário no final do código que você quer utilizar alguma operação de atualização da variável, senão o programa entra em um loop infinito, isso não precisa na próxima função que vamos apresentar. A forma que o while é apresentado no python é assim:

```
A = 10
B = 0
while(B < A):
        print('olá')
        B = B + 1
```

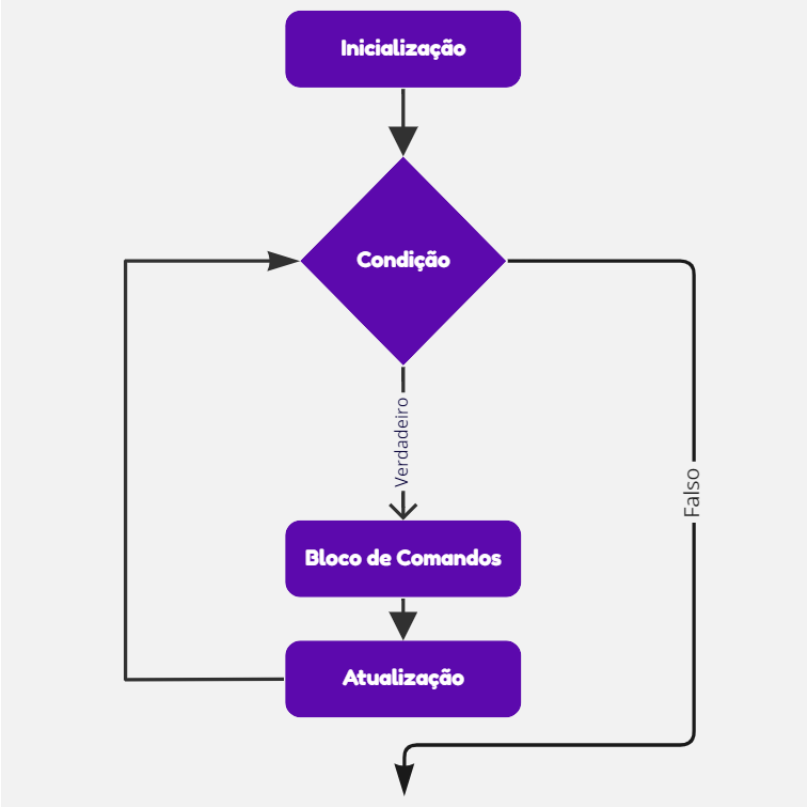
O que esse programa estar dizendo é que toda vez que B for menor que A, escreverá olá na tela, logo em seguida vai adicionar 1 no valor de B, esse seria a operação de atualização da variável já citado. Ou seja, quando o programa vai escrever olá até B ser maior que A, que são 10 vezes.

2.5 For

O for é uma estrutura de repetição bastante utilizada que lhe permite executar uma série de comandos várias vezes. O seu modo de funcionamento é bastante parecido com o while. Basicamente, podemos dividir o for em três partes, sendo a primeira a chamada de inicialização, a segunda é a condição e a terceira é a atualização. Descrevendo cada uma de forma mais detalhada, temos:

- **Inicialização:** Diferente do while, no for podemos declarar variáveis e/ou inicializar seus valores e fazemos isso na primeira parte do for. Perceba também que essa etapa é opcional;
- **Condição:** Essa parte é a condição que vai manter o for em loop e será verificada a cada iteração. Caso a condição seja verdadeira um bloco de instruções será executado e caso contrário o loop é interrompido;
- **Atualização:** E a ultima parte é a atualização, onde podemos atualizar o valor de variáveis, tanto as que foram declaradas e/ou inicializadas na Inicialização quanto qualquer outra variável.

Observe o fluxograma abaixo para você entender melhor o funcionamento do for:



Fluxograma do for.

A forma que seria representada o for no python seria:

```
A = (1, 2, 3, 4, 5)
for B in A:
        print(B)
```

Lembrando que no python para utilizar o for é necessário tem uma lista com várias variáveis, isso será melhor explorado no capítulo de vetores. Nesse programa ele tem a variável que vai ficar atualizando (B) e vai ficar verificar os valores contidos na lista A, dessa forma ela vai imprimir os valores de B, que segundo o loop, vai ser cada um dos termos da lista A.

2.6 Problemas

1) Faça um programa que leia uma senha e veja se essa senha é igual a “1234”. Mostre na tela se o valor inserido é correto ou incorreto.

2) Faça um programa que mostre todos os números pares entre 1 a 100.

3) Faça um programa que leia 6 valores (positivos e negativos, não pode ter 0) e mostre quantos números positivos existem dentre os 6.

4) Escreva um programa que leia as medidas dos lados de um triângulo e escreva se ele é equilátero, isósceles ou escaleno.

5) Escreva um programa para ler o ano de nascimento de uma pessoa e escrever uma mensagem que diga se ela poderá ou não votar este ano (considerar apenas o ano em que a pessoa nasceu).

6) Supondo que a população de um país A seja da ordem de 80000 habitantes com uma taxa anual de crescimento de 3% e que a população de B seja 200000 habitantes com uma taxa de crescimento de 1.5%. Faça um programa que calcule e escreva o número de anos necessários para que a população do país A ultrapasse ou iguale a população do país B, mantidas as taxas de crescimento.

7) Faça um programa que realize qualquer equação de segundo grau para qualquer coeficiente A, B e C. Mostre se a equação tem 1, 2 ou nenhuma solução, mostrando todas as soluções ou colocando uma mensagem dizendo que não existe soluções.

8) Faça um programa que peça dois números, base e expoente, calcule e mostre o primeiro número elevado ao segundo número. Não utilize a função de potência da linguagem.

9) 2520 é o menor número que pode ser dividido por todos os números de 1 até 10 sem deixar resto (ou seja, divisível por todos os números de 1 até 10). Faça um programa que mostre o menor número divisível por todos os números divisíveis por 1 até 20.

10) Se listarmos todos os números naturais abaixo de 10 que são múltiplos de 3 ou 5, obtemos 3, 5, 6 e 9. A soma desses múltiplos é 23. Faça um programa que mostre a soma de todos os múltiplos de 3 e 5 abaixo de 1000.

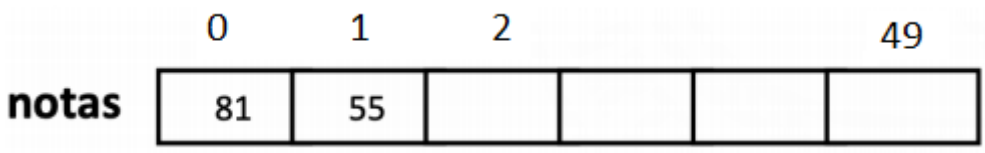
3. Vetores

Vetor, de maneira simples, nada mais é que um local para armazenar valores de um mesmo tipo em determinadas posições. Assim, os vetores podem ter um tamanho pequeno, ou gigantesco, tudo dependerá da necessidade do seu uso. Sua utilidade principal é a de armazenar mais de um valor de mesmo tipo, pois como visto anteriormente, as variáveis só podem receber um valor por vez, já os vetores podem receber vários ao mesmo tempo, tudo vai depender da dimensão dele. Abaixo, estão alguns exemplos de diferentes vetores, cada um com seu tipo de variável específica.

```
Exemplo 1:
Vetor inteiro = { 0, 1, 2, 3, 4, 5 }
Vetor float = {0.1, 1.2, 2.3, 3.4, 4.5, 5.6 }
Vetor char = { 'a', 'b', 'c', 'd', 'e', 'f'}
```

Repare que cada vetor possui um tipo específico de variável atrelada a si, e todas as células (posições) possuem o mesmo tipo de variável, essa é uma das principais características dos vetores, eles não possuem variáveis de tipos diferentes juntas. Algo importante de se comentar é que os vetores são contados de maneira diferente, pois a posição inicial é o zero e não um, ou seja, ele começa a ser contado a partir de zero.

Exemplo 2: Suponhamos que exista um vetor com notas de 50 alunos;



Exemplo 2.

Como mostra a figura, a nota do aluno 1 está na posição 0 (índice 0) do vetor, a do aluno 2 está na posição 1 (índice 1) e assim por diante.

3.1 Declaração de um vetor

Para declarar um vetor em Python é bem intuitivo e bem fácil se comparar com outras linguagens. Como vimos, um vetor é um conjunto com várias variáveis, portanto para que possamos declarar um vetor fazemos da seguinte maneira:

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

Nesse vetor que criamos colocamos uma lista de números de 1 a 5. Poderíamos também colocar um vetor com nomes, que ficaria

```
vetor = ['Chorão', 'Champignon', 'Thiago', 'Pinguim', 'Marcão']
```

O código acima atribui a vetor os nomes que foram colocados entre os travessões, como pode-se ver, não precisa declarar o tipo de variáveis, apenas o nome do vetor e o nomes das variáveis.

3.2 Manipulando vetores

Como já foi apresentado o conceito de vetores e como declará-los, vamos para a parte mais importante que é manipulá-los. Primeiramente é importante falar que é possível colocar o vetor na tela como vimos em uma variável qualquer, por exemplo, no código anterior, se utilizarmos a função "print" no vetor, desse modo:

```
vetor = ['Chorão', 'Champignon', 'Thiago', 'Pinguim', 'Marcão']
print(vetor)
```

No caso do código acima o que iria retornar pra gente seria todos os nomes dentro do vetor, que no caso é: ['Chorão', 'Champignon', 'Thiago', 'Pinguim', 'Marcão'].

Um comando importante de se falar aqui, e que é uma particularidade da linguagem Python, é o comando de você adicionar variáveis em um vetor:

```
vetor = ['Chorão', 'Champignon', 'Thiago', 'Pinguim', 'Marcão']
vetor.append('Heitor')
print(vetor)
```

No exemplo acima, a variável "Heitor" foi adicionado no vetor depois da variável "Marcão", retornando da seguinte maneira: Chorão, Champignon, Thiago, Pinguim, Marcão, Heitor. É possível também remover elementos da lista, apenas utilizando a função "remove" e entre parênteses número da posição da variável que você quer remover, exemplo:

```
vetor = ['Chorão', 'Champignon', 'Thiago', 'Pinguim', 'Marcão']
remove(0)
print(vetor)
```

Nesse exemplo acima removemos a variável da posição 0 do vetor, que é "chorão", ou seja esse código iria retornar: Champignon, Thiago, Pinguim, Marcão, Heitor. Com todas as funções apresentadas já existe um leque muito grande de coisas que pode-se fazer utilizando vetores, como apresentar apenas um elemento do vetor, que é apenas colocando entre travessões, exemplo: "print(vetor[0])", iria mostrar apenas a posição 0 do vetor.

3.3 Laços com vetores

Uma coisa que ajuda bastante a vida do programador na linguagem Python é a função "in", que é usada pra aplicar uma outra função apenas em um vetor. Exemplo:

```
vetor = ['Chorão', 'Champignon', 'Thiago', 'Pinguim', 'Marcão']
if 'Levir' in vetor:
        print('Ele está na banda.')
else:
        print('Ele não está na banda.')
```

No código acima ele procura a variável, do tipo String, "Levir" dentro do vetor (por isso in vetor) e se essa variável estiver, printa o que tá no if e se não tiver printa o que tá no else. Nesse caso iria retornar "ele não estar na banda" porque a variável "Levir" não estar no vetor. É possível também utilizar o comando "for" utilizando a função in, nesse caso o in iria falar que vetor iria aplicar o loop, que é o que a gente viu na aula passada.

3.4 Bubble sort

Vou falar agora de um algoritmo que não é importante para o Python (vocês vão entender no final o motivo), mas é importante entender ele para aprender como que um lista é ordenada, que é o algoritmo Bubble Sort. Esse algoritmo ele ordena uma lista de números (de forma crescente ou decrescente) de um forma prática mas pensando no computador é uma forma lenta. Esse algoritmo funciona da seguinte maneira, você compara um número da lista com todos os seus sucessores (todos os da frente dele) e verifica se é maior ou menor que ele (vai depender de que forma você vai querer ordenar a lista) e se for você troca eles de posição, isso resulta em ordenação da lista. Mas no Python isso tudo é ordenado apenas com um comando que é o "sort", exemplo:

```
vetor = [2, 3, 5, 4, 1]
vetor.sort()
print(vetor)
```

No exemplo acima iria retornar o vetor de forma ordenada: "1, 2, 3, 4, 5", a mesma coisa que o algoritmo do Bubble Sort iria fazer.

3.5 Problemas

1) Considere uma lista ordenada. Faça programas que realizam as tarefas abaixo:
a) Verificar se existe algum número x na lista cujo dobro 2x também está na lista.
b) Verificar se existe algum par de elementos na lista cuja soma é exatamente igual a x.

2) Agora considere uma lista desordenada. Faça um programa que move todas as ocorrências do número x para o início da lista.
Nota: Tome cuidado para não perder nenhum elemento.

3) Considere dessa vez duas listas ordenadas. Faça um programa que organiza todos os elementos das listas 1 e 2 em uma única lista ordenada.

4) Considere uma lista desordenada. Imagine que a variável x armazena um número qualquer. Faça um programa que move todos os elementos menores que x para o início da lista, e move todos os elementos maiores do que x para o final da lista. Por exemplo, se a lista contém os números:
52, 12, 78, 25, 44, 10, 98, 61, 82, 34
e x armazena o número 30, então uma solução possível seria:
12, 25, 10, 52, 78, 44, 98, 61, 82, 34

5) Se o maior elemento da lista ocorre duas vezes, o segundo maior elemento da lista tem o mesmo valor (é o mesmo número).
encontre o segundo maior valor da lista. Por exemplo, considere a seguinte lista:
22, 45, 27, 43, 45, 11, 12, 31, 45, 14
O procedimento deve retornar 43. Caso não haja um segundo maior valor, imprima uma mensagem dizendo que todos os elementos da lista são iguais e diga qual é esse elemento.

6) Imagine que você tem um vetor com elementos repetidos. Por exemplo:
1, 6, 3, 3, 5, 7, 2, 1, 5, 1
A tarefa consiste em procurar as duas primeiras ocorrências do número repetido mais repetido e mover a segunda ocorrência para a posição logo após a primeira (se no vetor tiver ao menos 2 ocorrências, é claro).
No exemplo acima, como elemento mais repetido é o 1, isso nos daria
1 1 3 3 5 7 2 6 5 1

7) Imagine que você tem um vetor com números inteiros armazenados na memória. Por exemplo:
9, 42, 21, 14, 25, 3, 19, 33, 45, 6
A tarefa consiste em imprimir todos os elementos entre 10 e 20, e contar a sua quantidade. No exemplo acima, isso nos daria
14, 19 : 2 elementos

8) Imagine que você tem um vetor com números inteiros armazenado na memória. Por exemplo:
17, 4, 32, 6, 51, 9, 89, 16, 1, 22
A tarefa consiste em contar a quantidade de elementos pares e contar a quantidade de elementos ímpares. No exemplo acima, isso nos daria:
5 pares, e 5 ímpares

9) Imagine que você tem um vetor com números inteiros armazenado na memória. Por exemplo:
31, 6, 53, 18, 21, 41, 64, 15, 2, 68
A tarefa consiste em localizar um elemento qualquer na lista e imprimir a sua posição se ele estiver lá, se ele não estiver imprima uma mensagem dizendo que esse

elemento não estar no vetor. No exemplo acima, se o elemento for 41, isso nos daria:
achei o 41 na posição 6
e se fosse 17, isso nos daria:
não encontrei o 17

10) A lista unimodal é uma lista onde os primeiro os elementos crescem e depois eles decrescem (ou vice-versa), exemplo:

lista unimodal

A tarefa consiste em Localizar o ponto em que a lista muda de direção.

4. Funções, ponteiros e recursividade

4.1 Funções

Por mais que esteja sendo apresentado como um conceito novo, vocês já trabalharam com funções, como a função print, que é um comando já existente no Python para que você possa colocar qualquer mensagem na tela. Da mesma forma que existe esse comando que já é algo embutido na linguagem, você pode criar suas próprias funções, ou comandos, para que eles sejam usados posteriormente no programa que você estar desenvolvendo.

Um bom programador sempre utilizará bastante das funções, porque elas além de facilitar o seu trabalho, simplifica bastante o seu código e evita você se perder nele, também facilita outras pessoas trabalharem nesse mesmo código, portanto é bastante importante o domínio de funções e que seja haja uma boa utilização desse recurso.

Bem, já que entendemos qual a finalidade das funções, vamos ver como aplicá-las:

```
def nome():
        print('olá, sou Levir')
nome()
nome()
nome()
nome()
```

Nesse exemplo acima criamos um programa que digita várias vezes uma mesma mensagem na tela, mas perceba que nesse caso eu não repeti várias vezes a função “print”, criei uma função que realiza essa operação e depois apenas utilizei o comando (o nome que coloquei para a função, que nesse caso é “nome”) e poupei um trabalho a mais que teria.

Esse exemplo é o caso mais simples de utilização de funções, existe também o caso que a função pode receber variáveis:

```
def divisao(n1, n2):
        return n1 / n2
num = divisao(8, 2)
print(num)
```

No caso acima temos o exemplo de uma função que realiza uma divisão, perceba que como utilizaremos variáveis (n1 e n2), elas foram colocadas dentro do parênteses em que a função foi declarada. Outro ponto importante de se ressaltar é a questão do “return”, ele significa o valor que vai ser retornado, ou seja, ele significa o valor que vai resultar quando eu utilizar aquela função, nesse exemplo eu estou retornando uma divisão de dois números, então toda vez que eu utilizar o comando “divisao” vai estar resultando na divisão desses dois números. Como vocês podem ver, no exemplo eu usei a variável “num” para receber a função, que teve os números 8 e 2 como entrada e retornou a divisão dessas entradas, nesse caso, quando eu rodar esse programa vai aparecer o resultado dessa divisão, que é 4.

4.2 Variáveis locais e globais

Esse tópico ele é mais simples e mais curto que o anterior, mas ele acaba sendo bem importante. De forma bem simples, as variáveis locais são as que vale apenas para uma função criada, já as globais são as que valem para todo o código. Usando o exemplo da função “divisao” que eu criei, os valores n1 e n2 são variáveis locais, ou seja, elas só podem ser trabalhadas dentro da função “divisao”, se por acaso eu utilizar essas variáveis em qualquer outra parte do código ou até mesmo em outra função, elas não vão ser reconhecidas porque são locais da função “divisao”. Agora se por exemplo eu declarasse n1 e n2 como sendo quaisquer valores (sem que sejam em função), eles acabam se tornando variáveis globais, você pode estar utilizando eles em qualquer ponto do código, sejam em variadas funções ou na função principal (em Python o conceito de função principal não é tão explorado já que por simplicidade da linguagem, acaba não sendo necessário, mas isso pode ser bem explorado em outras linguagens como C e C++).

4.3 Problemas

1) Escreva uma função que calcule e retorne a distância entre dois pontos (x1, y1) e (x2, y2).

2) Crie um programa que realize todas as quatro operações básicas com funções para cada uma delas. Os valores devem ser digitados pelo operador. Lembre-se que a divisão só ocorre quando o divisor não é zero. Mostre uma mensagem dizendo que a operação não existe, se for o caso.

3) Faça um programa que receba a resposta de três sensores, sendo um sensor de temperatura, um de umidade e pressão atmosférica. Suponha que os valores dados pelo sensor possuem unidades do Sistema Internacional de Unidades (SIU). O programa deve fazer a leitura por meio de três funções e o programa principal deve comparar os valores lidos com a temperatura de 27 ºC, umidade de 77 % e pressão atmosférica de 100 Pa. Se tais condições foram atendidas, deve-se emitir um alerta com "Condições

propícias para dormir", caso contrário, o programa não deverá emitir mensagens.

4) Escreva um programa, utilizando funções, para solicitar ao usuário o raio de uma esfera, e calcular o volume V da esfera usando uma função, e exibir o resultado.

$$V = \frac{4}{3} r^3$$

5) Fazer uma função para calcular e retornar o logaritmo em qualquer base. Faça ainda um programa que leia o logaritmando e a base.
Dica: lembre-se que o logaritmo de x em qualquer base b pode ser escrito em função da divisão do logaritmo de x numa base n qualquer pelo logaritmo de b na mesma base n, para qualquer n real.