

Aula 03 – Operadores em Python 3

Parte 1 – Revisão Teórica

A ATRIBUIÇÃO DE VALORES

A Atribuição de Valores é a passagem de informação a determinada variável. Toda variável, por sua definição, pode receber valores ou então, pode ter seu valor alterado.

Nesse primeiro momento, estudaremos a atribuição simples, isto é, a forma comum de atribuição.

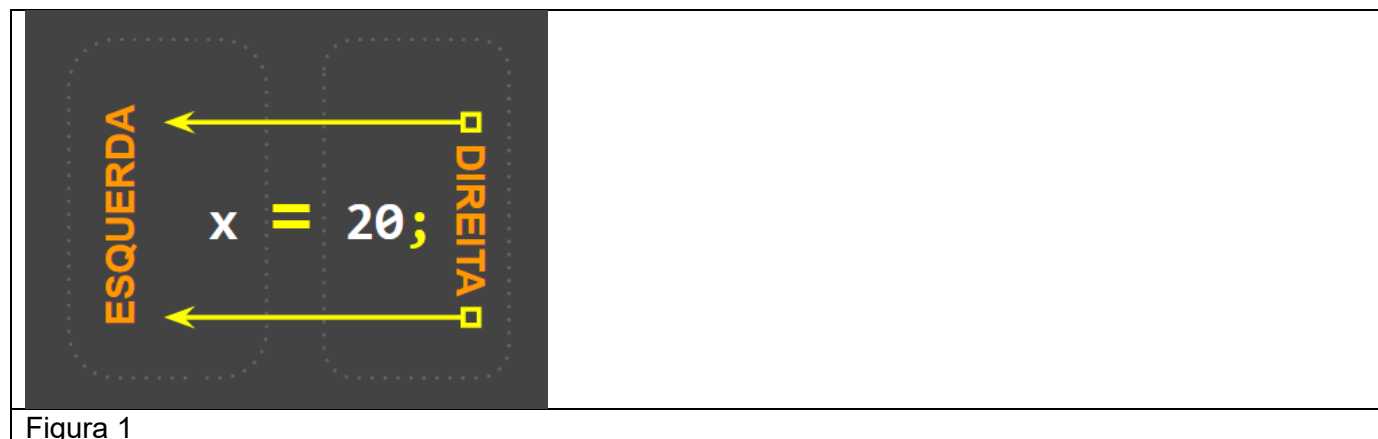


Figura 1

A linguagem Python tem definido que o sinal que conhecemos pelo nome de igual (=) será o operador de atribuição. Desta forma, iremos construir expressões compostas de 2 partes, ou melhor que estarão em um dos dois lados do operador de atribuição: do lado esquerdo e do lado direito.

A parte ao lado esquerdo do operador de atribuição **sempre** receberá o valor definido no lado direito do operador. Assim, é uma norma da linguagem que o lado esquerdo possua uma referência que possa ter seu valor alterado, enquanto que do lado direito, haja algum valor a ser atribuído à referência ao lado esquerdo.

```
<variável> [operador] <valor>
<variável>    =    <valor>
numero        =    10
```

Listagem 1

No trecho de código acima podemos ver como ocorre a passagem de valores e o uso do operador de atribuição na prática.

A notação para atribuição de valores define que a variável que receberá o valor estará do lado esquerdo do sinal de atribuição =, enquanto o valor a ser atribuído, estará do lado direito. Assim, **é correto afirmar** que a atribuição de valores sempre seguirá o sentido da direita para a esquerda.

No código a seguir, temos mais um exemplo onde declaramos variáveis e já atribuímos um valor às mesmas.

```
num = 10
txt = "texto"
```

Listagem 2

No código acima, declaramos 2 variáveis, a primeira de nome num e a segunda de nome txt. Ambas foram declaradas e receberam um valor inicial. O valor inicialmente atribuído irá definir o tipo da variável, de modo a impedir que a mesma receba valores de outro tipo durante toda a execução do Script.

ATRIBUIÇÃO SIMPLES A VÁRIAS VARIÁVEIS

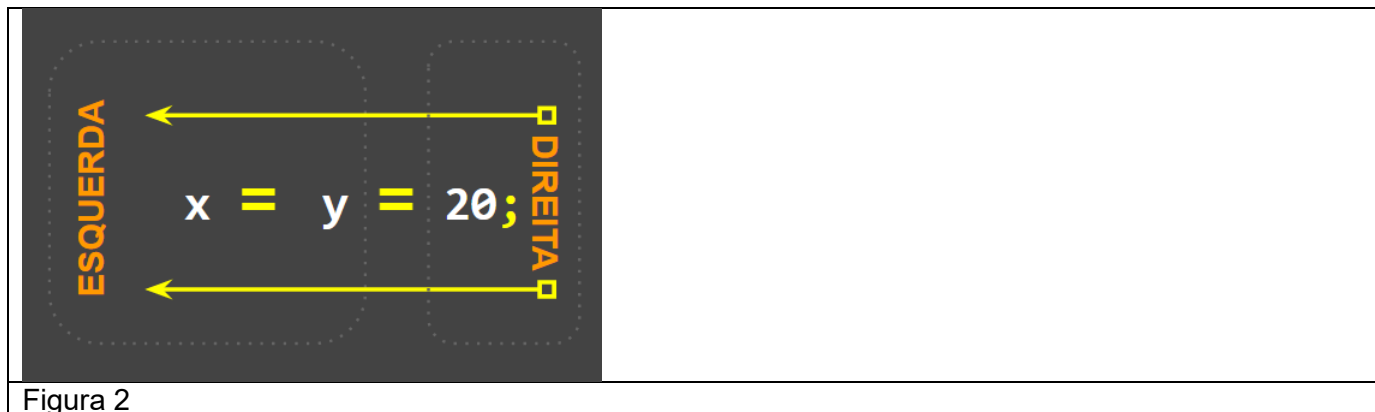


Figura 2

A notação de atribuição simples do Python nos permite atribuir a várias variáveis um mesmo valor numa mesma expressão. Para isso, devemos separar as variáveis que receberam o mesmo valor com vírgulas. No geral, as regras da atribuição estudadas funcionam da mesma forma.

No código abaixo, estamos realizando outra atribuição simples, porém, utilizando uma notação diferente.

```
a = 1  
b, c = 2
```

Listagem 3

No código acima, declaramos 3 variáveis. A primeira, chamamos de a e atribuímos o valor 1. A segunda e a terceira, chamamos de b e c, separando-as por vírgula e iniciando-as com o valor igual a 2.

Podemos iniciar mais de uma variável numa mesma atribuição, ou então, em outras palavras, podemos atribuir valores a várias variáveis ao mesmo tempo e a notação utilizada nessas 2 situações foi a mesma, salvo o uso da vírgula na separação da primeira e da segunda variável que receberão um novo valor.

ATRIBUIÇÃO MULTIPLA

A atribuição múltipla é a capacidade de atribuir um conjunto de valores a um conjunto de variáveis. Ambos conjuntos precisam, obrigatoriamente, possuir a mesma quantidade de elementos, isto é, a quantidade de variáveis a esquerda deve ter a mesma quantidade de valores a direita, como pode ser visto no código a seguir.

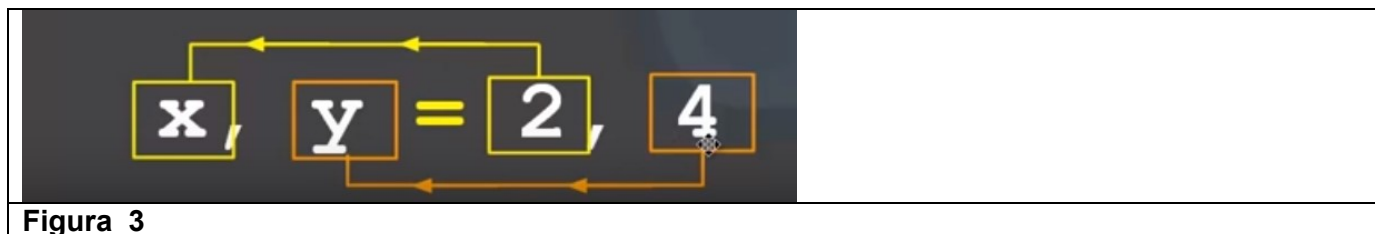


Figura 3

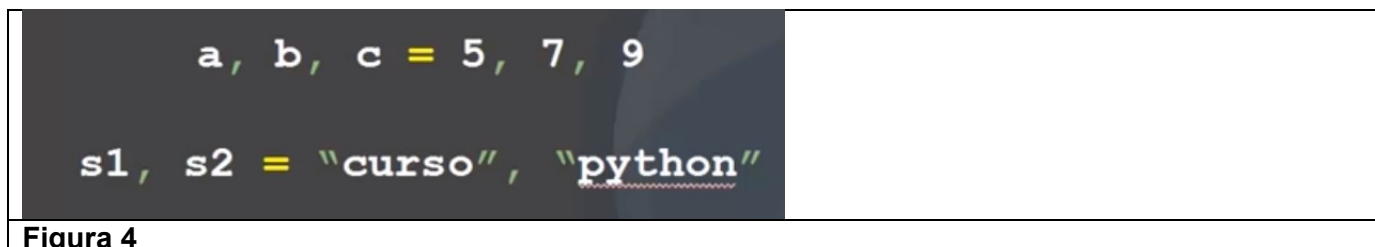


Figura 4

No código acima, estamos declarando e atribuindo valores a 2 variáveis. A primeira variável de nome **a**, está sendo atribuído o valor 1, e a segunda variável de nome **b**, está sendo atribuído o valor 2.

A ordem em que as variáveis estão dispostas a esquerda, receberá, na mesma ordem, os respectivos valores a direita.

Assim, sendo a variável **a** a primeira, e a variável **b** a segunda, estas receberam, respectivamente, o primeiro valor, e o segundo valor, respectivamente, definido no lado direito do operador de atribuição.

- a) Operadores Aritméticos
- b) Operadores Relacionais
- c) Operadores Relacionais Compostos
- d) Operadores Relacionais na Prática
- e) Operadores Lógicos
- f) Operadores de Atribuição
- g) Operador de Atribuição Múltipla
- h) Operador de Atribuição Compostos

INTRODUÇÃO ÀS OPERAÇÕES MATEMÁTICAS

O Python além de ser uma excelente linguagem para a construção de aplicações de forma geral, também é, uma excelente linguagem para ser utilizada junto com a matemática, como por exemplo, na geração de relatórios, gráficos, estatísticas e etc.

Atualmente, temos diversas bibliotecas matemáticas para serem utilizadas, e os recursos vão de simples análises e geração de gráficos a inteligência artificial, análise de sentimentos, estatísticas para compra e venda de ações, neurociência e muito, mas muitas outras áreas relacionadas a pesquisa e análise de informações.

OPERADORES MATEMÁTICOS

O Python pode ser utilizado como uma calculadora matemática avançada. Praticamente, todos os operadores aritméticos funcionam da mesma forma como os conhecemos da matemática elementar. Por exemplo, para trabalharmos com as 4 principais funções matemáticas, a soma, subtração, multiplicação e divisão, temos os operadores conforme tabela a seguir.

Operação	Operador
adição	+
subtração	-
multiplicação	*
divisão	/

Temos também, operadores para exponenciação, obtenção da parte inteira de uma divisão, extração do módulo da divisão, conforme pode ser visto na tabela a seguir:

Operação	Operador
exponenciação	**
parte inteira	//
módulo	%

O operador módulo % será estudado em detalhes em aulas futuras.



Comparando valores

Operador	Teste comparativo	Exemplo	
==	Igualdade	0 == 0 A == a	True False
!=	Desigualdade	0 != 1	True
>	Maior que	0 > 1	False
<	Menor que	0 < 1	True
>=	Maior que ou igual a	0 >= 0	True
<=	Menor que ou igual a	1 <= 0	False

Os caracteres maiúsculos A-Z têm valores de código ASCII 65-90 e os caracteres minúsculos a-z têm valores de código ASCII 97-122.

 @usandopython [Link do Tutorial na Bio](#) www.usandopy.com

Podemos elevar um número a outro através da utilização de 2 sinais de multiplicação seguidos **, isto é, o operador ** é o operador para exponenciação em Python. Também é possível obter a parte inteira da divisão, com o uso do sinal de divisão repetido, //.

A seguir, temos algumas operações matemáticas utilizando os operadores que acabamos de estudar e seus respectivos resultados:

```
a, b, c = 2,4,8
```

```
print("O valor da variavel a = " +str(a))
print("O valor da variavel b = " + str(b))
print("O valor da variavel c = " + str(c))
```

```
a, b, c = a**2, b**3, c**4
```

```
print("Resultados da Potenciação e Radiciação")
print("O valor da Potencia de 2 da variavel a = " +str(a))
```

```
print("O valor da Potencia de 3 da variavel b = " + str(b))
print("O valor da Potencia de 4 da variavel c = " + str(c))
print("O valor da Potencia de 4 da variavel c = " + str(c))

print("Resultados Radiciação")
d = a**(1/2)
print("O valor da Radiciação de 2 da variavel 4 = " +str(d))
e = b**(1/3)
print("O valor da Radiciação de 2 da variavel 64 = " +str(e))
f = c**(1/4)
print("O valor da Radiciação de 2 da variavel 4096 = " +str(f))
```

Listagem 4

Acabamos de utilizar 6 operadores matemáticas que estão definidos por padrão. Ainda que não tenha muita utilidade num primeiro momento, podemos alterar o funcionamento dos operadores, ou seja, podemos fazer com que o sinal de adição funcione da maneira como bem desejarmos.

Essa funcionalidade não possui utilidade quando estamos tratando de números inteiros, porém, é bastante útil, quando por exemplo, temos uma estrutura que seria interessante poder ser somada com outra. Logo, implementamos na classe dessa estrutura a funcionalidade de soma, subtração e etc.

EXEMPLO

```
x, y, z = 10, 50, 80

print("O valor da variavel x = " +str(x))
print("O valor da variavel y = " + str(y))
print("O valor da variavel z = " + str(z))

print(x+x)
print(x+(y+y))

print(x-x)
print(x*x-z)

print(x/5)
print(x/6)
print(x//6)
```

Listagem 5

OPERADORES RELACIONAIS ou OPERADORES COMPARATIVOS

Operador Relacional é todo operador que obtém a relação do membro a esquerda com o membro a sua direita.

É comum na programação a necessidade de conhecer a relação entre diversos operandos para que então, o nosso programa assuma determinada característica, ou invoque alguma funcionalidade. A linguagem Python trabalha com os operadores relacionais, também chamados de operadores comparativos, da mesma forma que a maioria das outras linguagens, tais como C, C++, Java, C# e etc.

TABELA DOS OPERADORES RELACIONAIS

Na tabela abaixo, temos os operadores relacionais disponibilizados pelo Python. Na continuação deste artigo, estudaremos cada operador isoladamente.

Descrição	Operador
Maior que	>
Menor que	<
Igual a	==
Diferente	!=
Maior ou igual a	>=
Menor ou igual a	<=

OPERADORES COMPOSTOS

A seguir, temos a lista com os 5 operadores compostos do Python:

OPERADOR	NOME
+=	mais igual
-=	menos igual
*=	vezes igual
/=	dividido igual
%=	módulo igual

Como podemos ver na tabela acima, os operadores compostos são formados pela junção do operador de atribuição com o [operadores aritméticos](#), onde o operador matemático precede o operador de atribuição. Uma maneira de facilitar o entendimento e memorização dos operadores compostos, é saber lê-los corretamente, por exemplo: o operador que soma e atribui é chamado de Mais Igual, isto é, o valor a esquerda do sinal de atribuição será somado com o valor a direita e o resultado dessa operação, será atribuído à variável a esquerda do sinal de atribuição.

A seguir, um trecho de código em que é feito uma operação com cada um dos 5 operadores de atribuição compostos. A direita de cada operação há o respectivo resultado.

```
num, num2, num3,num4, num5, num6 = 10, 10 ,10 , 10 , 10,10

print("Parte I: valor inicial de num = " +str(num))
num += 1
print("Parte II: valor da variavel num + 1 = %i " %num)

num2 -= 1
print("Parte III: valor da variavel num2 - 1 = %i " %num2)

num3 *= 2
print("Parte IV: valor da variavel num3 * 2 = %i " %num3)

num4 /= 3
print("Parte V: valor da variavel num4 / 3 = ",num4)

num5 //= 3
print("Parte VI: valor da variavel num5 // 3 = %i " %num5)
```

Listagem 13

O mais importante a saber dessa aula, talvez seja, como ler cada operador e sempre lembrar, que antes de qualquer valor ser atribuído, é necessário resolver a operação a direita do sinal de atribuição.

EXEMPLO

```
a, b, c = 2,4,8

print("O valor da variavel a = " +str(a))
print("O valor da variavel b = " + str(b))
print("O valor da variavel c = " + str(c))

a, b, c = a**2, b**3, c**4

print("O valor da Potencia de 2 da variavel a = " +str(a))
print("O valor da Potencia de 3 da variavel b = " + str(b))
print("O valor da Potencia de 4 da variavel c = " + str(c))
```

Listagem 14

RELAÇÕES INTERESSANTES DO PYTHON

OPERADORES RELACIONAIS SIMPLES

Existem 3 relações possíveis entre 2 operandos, são elas:

Descrição	Operador
Maior que	>
Menor que	<
Igual a	==

Para obtermos a relação entre 2 membros, temos que utilizar a seguinte estrutura:

<membro a esquerda> OPERADOR <membro a direita>

É importante observar que a inversão dos membros ocasiona na inversão do resultado da expressão, isto é, se o membro que estiver a esquerda for para a direita e vice-e-versa, a relação entre eles será o contrário.

```
print( 1 > 2 )
print( "a" > "b" )
print( 5 < 10 )
print( 200 == 200 )
```

Listagem 25

No código acima, realizamos algumas operações entre números e caracteres, sempre utilizando as 3 comparações simples. Temos que, ou o operador a esquerda será maior, ou menor, ou então, igual ao operador a sua direita.

OPERADORES RELACIONAIS COMPOSTOS

A linguagem Python disponibiliza mais 2 operadores relacionais que verificam 2 relações entre os membros. É importante observar que essas relações podem ser obtidas com o uso de **conectores lógicos**, isto é, com a utilização dos conectores **and** e **or**.

Descrição	Operador
Maior ou igual a	<code>>=</code>
Menor ou igual a	<code><=</code>

Com os **Operadores Relacionais Compostos** podemos obter a relação se o operando a esquerda é maior ou igual do que o operando a direita, ou então, se o operando a esquerda, é menor ou igual do que o operando a direita.

O código a seguir imprime no Console os valores lógicos de 2 expressões.

```
print("1 >= 1: ", 1 >= 1 )  
print("1 <= 1: ", 1 <= 1 )
```

Listagem 26

No trecho de código acima, estamos verificando se os membros a esquerda são maiores ou menores do que os membros a sua direita.

Operadores de Atribuição Compostos

Os **Operadores de Atribuição Compostos** realizam uma operação e em seguida, atribuem o resultado da operação para a variável que está a esquerda do operador de atribuição.

Estudamos a atribuição simples em aulas passadas e aprendemos as 2 partes que constituem a estrutura de atribuição da linguagem, bem como o seu funcionamento de maneira geral.

Nesta aula continuaremos o estudo da atribuição, porém, estudaremos os operadores de atribuição compostos, isto é, os operadores que além de atribuírem o valor a sua direita à variável a sua esquerda, realizam antes alguma operação.

É comum precisarmos realizar alguma operação, geralmente matemática, e atribuir o resultado a alguma variável. Podemos resolver esse problema realizando a operação sobre determinado valor e em seguida, atribuir o resultado a alguma variável. No entanto, essa é uma situação tão corriqueira que, criou-se operadores que, primeiro, realizam uma operação e em seguida, atribuem o resultado à variável a esquerda.

Atualmente há 5 operadores compostos disponibilizados pela linguagem Python. Os 4 primeiros para as operações matemáticas fundamentais, soma, subtração, multiplicação e divisão. O quinto e último é para a operação módulo, isto é, o operador que retorna o resto de uma divisão.

PRIMEIRA RELAÇÃO

A primeira relação interessante, não é bem uma relação, mas sim, uma maneira de verificarmos se uma expressão é ou não verdadeira. Temos que ter em mente que o Python, SEMPRE verifica se determinada expressão é verdadeira e por consequência disso, podemos omitir a comparação com *True*.

Por exemplo, se quisermos saber se 2 membros são iguais, não precisamos compará-los a *True*, veja a expressão a seguir:

```
print( 1 == 1 ) #essa expressão equivale a expressão seguinte  
print( (1 == 1) == True )
```

Listagem 27

Na primeira operação, estamos verificando se o número 1 é igual a ele mesmo, logo, será impresso no Console, **True** ou **False**. Em seguida, repetimos a linha anterior, porém, comparamos o resultado da expressão `1 == 1` com o valor **True**.

Ainda que não haja problema na forma que trabalhamos na segunda operação, estamos sendo redundantes, haja vista que **toda expressão** será automaticamente comparada a **True** e por isso, podemos omitir essa comparação.

SEGUNDA RELAÇÃO

Uma expressão é verdadeira quando o resultado da expressão for igual a **True** ou então, igual a número inteiro 1. Logo, temos que as expressões no bloco de código a seguir, estão realizando a mesma comparação:

```
print( (1 == 1) == 1 )
print( (1 == 1) == True )
print( (1 > 1) == 0 )
print( (1 > 1) == False )

print( 1 == True )
print( 0 == False )
```

Listagem 28

Todas as expressões imprimirão no Console o valor **True**, então, é importante que saibamos que as palavras-chaves **True** e **False** na verdade, são constantes que armazenam os número 1 e 0 respectivamente.

INTRODUÇÃO PARA A TOMADA DE DECISÃO (Desvio condicional Simples) (if... Elif.... Else)

A **Tomada de Decisão** também chamado de **Desvio Condicional** ou então, a verificação de expressões é, uma prática comum no desenvolvimento de software. Com essa estrutura somos capazes de verificar expressões e assim, decidir qual **Bloco de Instrução** deve ser executado.

As expressões definidas no cabeçalho da estrutura if utilizam os operadores relacionais na construção das expressões, seja de maneira **implícita** ou **explícita**.

É importante observar que outras estruturas da linguagem trabalham de maneira similar ao que temos com as **Tomadas de Decisão**. Ou seja, há algumas estruturas em que uma situação será verificada e, em seguida, mediante ao valor retornado na avaliação, será definido o bloco de código a ser executado.

O entendimento da tomada de decisão é necessário para o bom estudo de toda a linguagem, até porque, essa é, a estrutura mais importante na construção de softwares.

GRÁFICO DA ESTRUTURA

A figura 5, temos uma representação do caso de uso da estrutura de tomada de decisão.

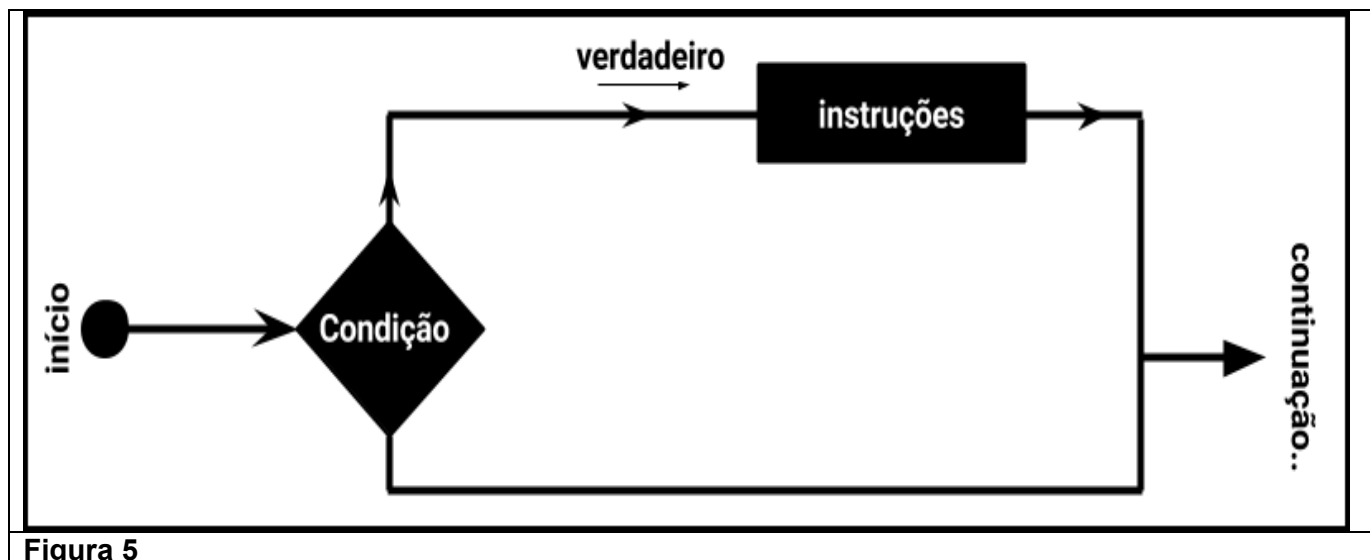


Figura 5

Na imagem acima, estamos, inicialmente, supondo que a execução da programação ocorra no sentido da esquerda para a direita. Assim, representamos com um círculo preto o início da execução.

Feito isso, chegamos na tomada de decisão propriamente dita. Nesta, será verificada a expressão contida no cabeçalho da estrutura if e, caso o valor avaliado seja verdadeiro, o bloco de instrução é executado e na sequência o programa continua sua execução normalmente.

Do contrário, ou seja, caso o valor avaliado no cabeçalho da estrutura if for false, nada ocorrerá, como está representado na imagem por uma linha que liga a estrutura if com a parte em que ocorre a continuação da execução normal das instruções definidas.

A INSTRUÇÃO if

A palavra if, do Inglês, significa SE. Com a estrutura if podemos determinar o que deve acontecer mediante o valor lógico retornado, isto é, **SE** for verdadeiro [True], faça isso, **SE** for falso [False], faça aquilo.

Analisemos o exemplo a seguir em que utilizamos o formato da instrução if, porém, utilizando a Língua Portuguesa:

UM CASO REAL UTILIZANDO A ESTRUTURA if

Agora, vamos fazer um exemplo onde verificaremos se o número digitado por um usuário é maior do que zero. Caso seja, o bloco de instrução [verdadeiro] a seguir da estrutura será executado.

```

num = int(input("Digite um número: "))

if(num > 0):
    print("O número digitado é maior do que 0", num)
  
```

Listagem 15

No exemplo acima, utilizamos a função input() para imprimirmos uma mensagem na tela e em seguida colocar o Console no modo de edição, isto é, colocar o Console de tal forma que seja possível ao nosso usuário digitar e pressionar Enter.

Após o usuário digitar um valor e pressionar a tecla Enter, o valor digitado é passado para a variável num. Em seguida, o valor contido na variável num é comparada ao número 0. Caso o valor de num seja maior do que zero, o bloco de instrução verdadeiro será executado, do contrário, nada acontecerá.

É importante observar que caso nada seja digitado, ou então, não seja digitado um número válido, uma exceção é levantada e a execução do programa é interrompida.

No código acima, estamos verificando se a expressão entre parêntesis é verdadeira. Em seguida, colocamos o sinal de 2 pontos, dizendo para o Python que ali finaliza o cabeçalho da estrutura if. A linha seguinte é o início do **Bloco de Instrução** e será executado nas situações em que a expressão for verdadeira. Todo bloco de instrução que está subordinado a uma estrutura, estará, obrigatoriamente, indentado um nível a frente da estrutura subordinada.

Nesse primeiro momento, temos que entender que se a expressão definida no cabeçalho da estrutura if [entre parêntesis], for verdadeira, tudo que estiver nas linhas seguintes e com a indentação avançada será executado. A instrução `faça_isso()` é a primeira instrução do **bloco de instrução** que é executado todas as vezes que a expressão definida e avaliada no cabeçalho da estrutura if é verdadeira.

Caso a expressão contida entre parêntesis não seja verdadeiro, na situação acima, nada acontecerá e a execução das linhas seguintes a tomada de decisão e que estão com a indentação avançada, não será executado. Entenda por **linhas seguintes** todas as instruções com o nível de indentação igual ao da instrução if.

INTRODUÇÃO À ESTRUTURA else

A **Tomada de Decisão** é composta por uma expressão a ser avaliada e o que deve acontecer caso a expressão seja verdadeira ou então, caso a expressão seja falsa. Por padrão, o bloco de instrução que estiver abaixo da instrução if será executado quando a expressão contida na estrutura if for verdadeira.

A estrutura else por sua vez, é uma instrução complementar da estrutura if. A palavra else do Inglês, significa SENÃO e com esta, definiremos o bloco de instrução a ser executado todas as vezes que a expressão definida for igual a falso.

ANÁLISE GRÁFICA

Na imagem a seguir temos a representação gráfica da tomada de decisão em que é definido o bloco de instrução que é executado caso o valor da expressão seja verdadeiro, ou então, o bloco de instrução que será executado caso o valor da expressão seja igual a falso.

O gráfico a seguir é bastante semelhante ao estudado na aula anterior, no caso, quando estudamos a tomada de decisão a utilização do bloco de instrução a ser executado caso o valor da expressão seja igual a falso.

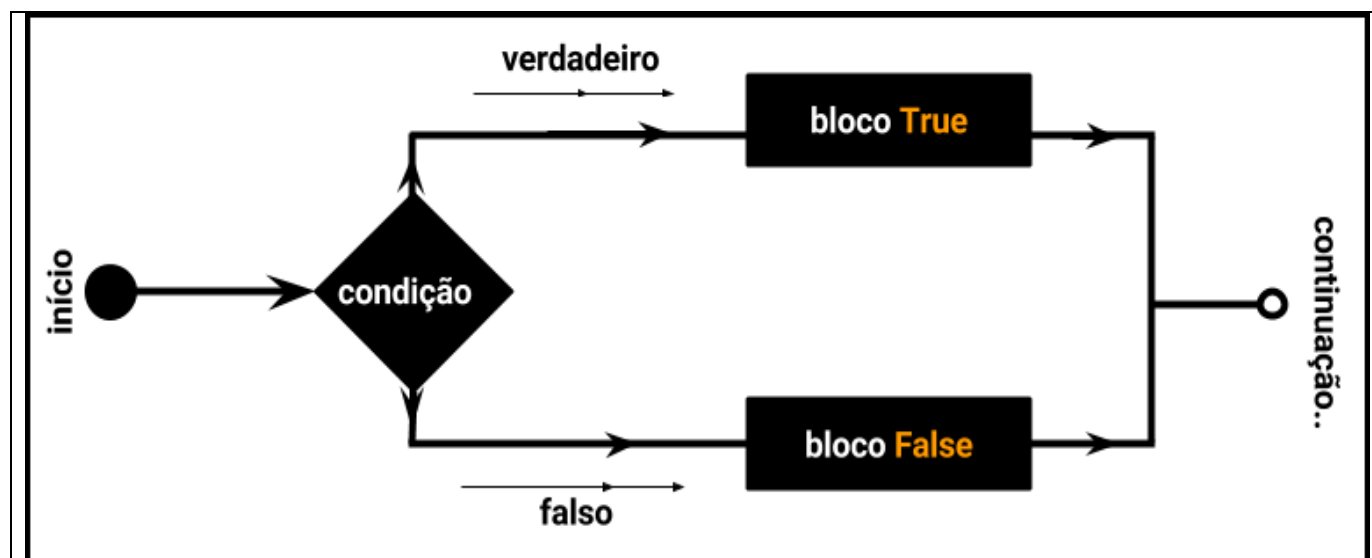


Figura 6

Na figura 6 é assumido que a execução do programa ocorre da esquerda para a direita. Assim, temos no extremo esquerdo um círculo preto que representa o início, ou então, as instruções que foram executadas anteriormente.

Feito isso, chegamos na tomada de decisão, que é representado por um losango. O losango representa as 2 situações possíveis na avaliação de qualquer expressão, isto é, o resultado por ser verdadeiro ou falso.

Caso seja verdadeiro, a execução está demonstrada na parte superior, onde temos a execução de um bloco de instrução e na sequência, o programa continua sua execução normalmente.

Na parte inferior, temos a execução do bloco de instrução para caso o valor da expressão seja falso. Como pode ser visto, um outro bloco de instrução é executado a a execução do nosso programa continua normalmente.

O bloco que estiver na linha seguinte a instrução if será o bloco de instrução a ser executados todas as vezes que a condição for verdadeira, enquanto que o bloco que estiver na linha seguinte a instrução else será executado todas as vezes que a expressão for falsa.

```
if(True):  
    print("")  
    if(True):  
        print("")  
    else:  
        print("")  
else:  
    print("")
```

```
if(True):  
    print("")  
    if(True):  
        print("")  
    else:  
        print("")  
else:  
    print("")
```

```
if(True):  
    print("")  
    if(True):  
        print("")  
    else:  
        print("")  
else:  
    print("")
```

Figura 7

A ESTRUTURA else

A instrução else é uma instrução dependente, isto é, uma instrução que não pode ser utilizada sozinha. A seguir, temos um exemplo, utilizando a instrução if junto a instrução else, porém, com as instruções traduzidas para o Português.

```
SE( verdadeiro )
    faça_isso()
SENÃO
    faça_aquilo()
```

Listagem 16

Exemplo que determinar se o número digitado pelo usuário é maior ou menor que 0 (zero)

```
num = int(input("Digite um número: "))

if(num > 0):
    print("O número digitado = "+ str(num) + " é maior do que 0")
else:
    print("O número digitado = "+ str(num) + " é menor do que 0")
```

Listagem 17

EXEMPLO

Determine se o número digitado pelo usuário é par ou ímpar. Dica, utilize a estrutura de desvio condicional (if...else)

```
num = int(input("Digite um número : "))

print("Programa que determina se o numero digitado é para ou impar")
if(not num % 2):
    print("O número digitado é par." ,num)
else:
    print("O número digitado é ímpar." ,num)
```

Listagem 18

No pseudocodigo abaixo, testamos se o nome “x” contém a string “Liquido”

```
if
    Se for verdadeiro, o código imprime “Menos de 100 Graus centigrados”

elif
    Senão, se “x” não é “liquido”, mas “vapor”, o código imprimi “Mais de 100 Graus centigrados”

else
    Caso “x” não seja nem “liquido” nem “vapor”, o resultado é “Menos de 0 Graus centigrados”
```

Listagem 19

Exemplo que determinar qual é estado (liquido, sólido ou vapor) da variável de entrada

```
x = input("Digite um dos valores: liquido, solido ou vapor = ")

print("")
print("Valor digitado pelo usuario = " +x)

if (x == "liquido"):
    print("Menos de 100 Graus centigrados")
```

```
elif (x == "vapor"):
    print("Mais de 100 Graus centigrados")

else:
    print ("Menos de 0 Graus centigrados")
```

Listagem 20

Atenção para a indentação: tudo que desejamos que seja feito caso *x* seja "líquido" está alinhado. O mesmo com *elif* e *else*.

No código acima, estamos verificando uma condição e, caso o resultado seja verdadeiro, o bloco de instrução na linha seguinte a instrução *SE* é executada.

Senão, caso o valor da expressão definida no cabeçalho da instrução *if* não seja verdadeiro, o bloco de instrução definido na linha seguinte a instrução *SENÃO* é o que executado.

```
nota1 = float(input("Informe a primeira nota = "))
nota2 = float(input("Informe a segunda nota = "))

media = (nota1 + nota2) / 2.0

print("A media do aluno é", media)
if (media == 10):
    print("Parabens, vc foi aprovado com Distinção")
elif (media >= 7):
    print("Vc está aprovado")
else:
    print("Vc está Reprovado")
```

Listagem 21

EXEMPLO

```
idade = int(input("Informe a sua idade: "))

if(idade<=0):
    print("A sua idade não pode ser 0 ou menor do que 0!" , idade)
elif (idade>120):
    print("A sua idade não pode ser superior a 120 ano!", idade)
elif (idade>=16) and (idade<120):
    print("Parabens, vc pode votar!", idade)
elif(idade<16) :
    print("Você precisa ter mais do que 18 anos!", idade)
```

Listagem 22

EXEMPLO

```
acao = int(input("Digite '1' para sim e digite '2' para não: "))
```

```
if(acao==1):
    print("Você disse que sim!")
else:
    if(acao==2):
        print("Você disse que não!")
    else:
        print("O número digitado não é '1' e nem '2'!!!")
```

Listagem 22

TERCEIRA RELAÇÃO

Se verificarmos uma expressão composta por uma String, temos que o valor retornado será *True*, caso a mesma contenha 1 ou mais caracteres, do contrário, o valor retornado será *False*.

```
x = ""
if(x):
    print("O valor de x é igual a vazio")
else:
    print("A variável 'x' contém caracteres ")
```

Listagem 23

No código acima, declaramos a variável de nome *x* e em seguida utilizamos a variável "*x*" isoladamente numa tomada de decisão. Caso a variável *x* contenha um ou mais caracteres, o bloco de código da estrutura *if* será executado. Do contrário, o bloco da estrutura *else* será executado.

EXEMPLO

```
a=5
if(a==5):
    print("O valor da variavel a =" +str(a) + " é Verdadeiro = True" , type (a))
elif("dd"):
    print(" O Valor é DD")
else:
    print("Nenhuma das condições anteriores")
```

Listagem 24

Match Case no Python 3.10

Caso vc conheça um pouco de programação vai achar essa estrutura bem similar ao **Switch Case** por exemplo.

Você certamente já ouviu falar em *If* e *Else* no Python e provavelmente já teve algum código que foi escrever que você colocou vários deles para pegar todas as possibilidades de uma variável ou algo do tipo.

Como por exemplo para pegar um valor e identificar qual o dia da semana esse valor corresponde. Só que ao utilizar o Match Case nós vamos ter casos no Python, como assim? Vamos fazer basicamente ela estrutura, mas vamos escrever *Case* no lugar do *If* e *Else*.

Então vamos ter uma variável e caso o valor dela seja 1 por exemplo vamos retornar domingo e assim por diante.

```
print("Programa que converte dias das semana numerico em texto")
#numero_dia =0
numero_dia = int(input("Informe um numero para saber o dia da semana (1, 2, 3, 4, 5, 6, 7): "))
```

```

texto ="Um bom dia para aprender Python"

print("A variavel numero_dia é do tipo = ",type(numero_dia))

def dia_da_semana(numero_dia):
    if (numero_dia == 1):
        print("Domingo. " + texto)
    elif (numero_dia == 2):
        print ("Segunda feira. " + texto)
    elif (numero_dia == 3):
        print ("Terça feira. " + texto)
    elif (numero_dia == 4):
        print ("Quarta feira. " + texto)
    elif (numero_dia == 5):
        print ("Quinta feira. " + texto)
    elif (numero_dia == 6):
        print ("Sexta feira. " + texto)
    elif (numero_dia == 7):
        print ("Sabado. " + texto)
    else:
        print ("Valor invalido. Favor digitar numeros entre o intervalo (1, 2, 3, 4, 5, 6, 7)")
dia_da_semana(numero_dia)

print("Fim do programa")

```

Listagem 25

Vai facilitar tanto a sua escrita, pois vai diminuir a repetição das estruturas condicionais e vai deixar o código mais fácil de ser entendido.

```

dia = int(input("Digite um numero inteiro entre '1 e 7': "))
mensagem = "Hoje é um bom dia para aprender Python"

match dia:
    case 1:
        print(" Domingo. " + mensagem)
    case 2:
        print(" 2da Feira. " + mensagem)
    case 3:
        print(" 3ça Feira. " + mensagem)
    case 4:
        print(" 4ta Feira. " + mensagem)
    case 5:
        print(" 5ta Feira. " + mensagem)
    case 6:
        print(" 6ta Feira. " + mensagem)
    case 7:
        print(" Sabado. " + mensagem)
    case _:
        print(" Dia invalido")

```

Listagem 26

```

print("Menu, opções (1, 2, 3 e 4)")
escolha = int(input('Digite a sua escolha : ')) # idade = -25

match escolha:
    case 1:

```



```

    print("Sua escolha foi %i" %escolha)
case 2:
    print("Sua escolha foi %i" %escolha)
case 3:
    print("Sua escolha foi %i" %escolha)
case 4:
    print("Sua escolha foi %i" %escolha)
case _:
    print('Escolha invalida')

print("Fim do programa")

```

Listagem 27

```

op = int(input("Digite o código do produto: "))

match op:
    case 1:
        print("\nAlimento não perecível!")
    case (2 | 3 | 4):
        print("\nAlimento perecível!")
    case (5 | 6):
        print("\nVestuário!")
    case 7:
        print("\nHigiene pessoal!")
    case (8 | 9 | 10 | 11 | 12 | 13 | 14 | 15):
        print("\nLimpeza e utensílios domésticos!")
    case _:
        print("\nOpção inválida!")

```

Listagem 28

Agora você já pode utilizar essa estrutura para facilitar seus códigos e evitar a repetição das estruturas If e Else.

BLOCOS DE CÓDIGO

Bloco de código é uma ou um conjunto de instrução que estejam numa mesma distância da margem esquerda. A seguir, temos um trecho de código Python que mostra a utilização de 2 blocos. Estes foram definidos pela quantidade de espaços, isto é, pela distância da margem esquerda.

```

print(nível 1)#primeiro nível hierárquico

if(True):
    print(nível 2)#segundo nível hierárquico

```

Este é um bloco de comentários e essa é a primeira linha agora, estamos na segunda linha de comentário(s)

É comum utilizarmos 2 espaços ou então, 4 espaços, ou mesmo 1 tabulação ao lado esquerdo da instrução para assim definir, em qual bloco a instrução está contida. O primeiro nível é o nível 0, ou seja, o nível que não contém espaçamento. A linguagem não nos obriga a utilizar uma determinada quantidade de espaçamentos, ou então, tabulações. Porém, se utilizarmos 4 espaços para definir o primeiro bloco, o interpretador assumirá que as próximas instruções estão indentadas com uso de múltiplos de 4.

A recomendação é que utilizemos, ou 1 tabulação, ou então, 4 espaços.

Ao invés de trabalharmos com quantidades de espaços, podemos utilizar uma determinada quantidade de tabulações. O primeiro nível hierárquico seria o nível 0, isto é, instruções que não possuem tabulações a sua esquerda. O segundo nível utilizaria uma única tabulação, o terceiro nível utilizaria 2 tabulações e assim sucessivamente.

O mais importante é sabermos que em Python, o espaçamento não é facultativo, isto é, não o colocamos se assim desejarmos, mas sim, somos obrigados a trabalharmos com algum sistema de espaçamento a fim de definirmos blocos isolados de códigos.

Os benefícios do uso da tabulação são percebidos rapidamente, isso porque, um código em Python sempre estará organizado, do contrário o mesmo não funcionará. E não somente isso, código indentados tornam-se mais legíveis e é um padrão utilizado por praticamente todos os programadores, indiferente da linguagem em que estes estejam trabalhando.

Glossário

atribuição (*assignment*)

Comando que atribui um valor a uma variável.

avaliar (*evaluate*)

Simplificar uma expressão através da realização de operações, para produzir um valor único.

comando (*statement*)

Trecho de código que representa uma instrução ou ação. Até agora, os comandos vistos foram de atribuição e exibição.

comentário (*comment*)

Informação em um programa dirigida a outros programadores (ou qualquer pessoa que esteja lendo o código fonte) e que não tem efeito na execução do programa.

composição (*composition*)

Habilidade de combinar expressões e comandos simples em expressões e comandos compostos, de forma a representar operações complexas de forma concisa.

concatenar (*concatenate*)

Juntar dois operandos lado a lado.

diagrama de estado (*state diagram*)

Representação gráfica de um conjunto de variáveis e os valores aos quais elas se referem.

divisão inteira (*integer division*)

Operação que divide um inteiro por outro e resulta em um inteiro. A divisão inteira resulta no número de vezes que o numerador é divisível pelo denominador e descarta qualquer resto.

expressão (*expression*)

Combinação de variáveis, operadores e valores, que representa um resultado único.

operando (*operand*)

Um dos valores sobre o qual o operador opera.

operador (*operator*)

Símbolo especial que representa uma computação simples, como adição, multiplicação ou concatenação de strings.

palavra-chave (*keyword*)

Palavra reservada usada pelo compilador/interpretador para analisar o programa; você não pode usar palavras-chave como `if`, `def`, e `while` como nomes de variáveis.

ponto-flutuante (*floating-point*)

Formato para representar números que possuem partes fracionárias.

regras de precedência (*rules of precedence*)

O conjunto de regras que governa a ordem em que expressões envolvendo múltiplos operadores e operandos são avaliadas.

tipo (type)

Um conjunto de valores. O tipo de um valor determina como ele pode ser usado em expressões. Até agora, os tipos vistos são: inteiros (tipo `int`), números em ponto-flutuante (tipo `float`) e strings (tipo `string`).

valor (value)

Um número ou string (ou outra coisa que ainda vamos conhecer) que pode ser atribuída a uma variável ou computada em uma expressão.

variável (variable)

Nome que se refere a um valor.

Bibliografia

Python, Escreva seus primeiros programas, Felipe Cruz, Casa do Código.

Curso Introdutório de Python, Grupy-Sanca, 09 de maio de 2019

Tutorial Python, Release 2.4.2. Guido van Rossum, Fred L. Drake, Jr., editor. Tradução: Python Brasil

Curso Python 3. Juracy Filho e Leonardo Leitão. Cod3r.com.br

https://aprendacompy.readthedocs.io/pt/latest/capitulo_01.html

<https://www.devmedia.com.br/python-trabalhando-com-variaveis/38644>

<https://cadernodelaboratorio.com.br/2017/12/01/variaveis-em-python3/>

Parte 2 – Prática (mão na massa)!!!

Para as atividades práticas vamos utilizar o VS Code, que deve estar previamente configurado para executar o python)

Para a criação dos arquivos (com extensão .py) vamos utilizar o editor de texto

Para executar, deve utilizar o interpretador do VS Code, temos 2 opções

- a) Apertar o Botão direito e selecionar a opção **RUN PYTHON TERMINAL**
- b) No menu superior, selecionar Debug -> **START WITHOUT DEBUGGING**

Para limpar o console, utilizar o comando **clear** ou **cls**

2) Criação do Projeto no VS CODE

- 1) Para isto crie uma pasta no seu pendrive com o nome **IPE-Python-2023**.
- 2) Dentro do **IPE-Python-2023**, criar a pasta **Aula03-Python**
- 3) Utilizando o VS CODE acesse a pasta criada (**IPE-Python-2023/ Aula03-Python**)
- 4) Para isso, acione a opção **File > New > File e crie os seguintes** (exemplo30.py, exemplo31.py, exemplo32.py, exemplo33.py....exemplo39.py), para melhor organização (**BOA PRÁTICA!!!**)

Exemplo 30

PASTA: IPE-Python-2023/ Aula03-Python
arquivo: exemplo30.py

```

1  x, y, z = 10, 50, 80
2
3  print("O valor da variavel x = " +str(x))
4  print("O valor da variavel y = " + str(y))
5  print("O valor da variavel z = " + str(z))
6
7  print(x+x)
8  print(x+(y+y))
9
10 print(x-x)
11 print(x*x-z)
12
13 print(x/5)
14 print(x/6)
15 print(x//6)
16

```

```

PS G:\IPE-Python2019> & C:/Users/SER...al/Programs/Python/Python37-32/python.exe g:/IPE-Python2019/Aula03-Python/exemplo30.py
O valor da variavel x = 10
O valor da variavel y = 50
O valor da variavel z = 80
20
110
0
20
2.0
1.6666666666666667
1

```

Figura A: exemplo30.py

Exemplo 31

PASTA: IPE-Python-2023/ Aula03-Python
arquivo: exemplo31.py

```

num, num2, num3, num4, num5, num6 = 10, 10, 10, 10, 10, 10

print("Parte I: valor inicial de num = " +str(num))
num += 1
print("Parte II: valor da variavel num + 1 = %i " %num)

num2 -= 1
print("Parte III: valor da variavel num2 - 1 = %i " %num2)

num3 *= 2
print("Parte IV: valor da variavel num3 * 2 = %i " %num3)

num4 /= 3
print("Parte V: valor da variavel num4 / 3 = ", num4)

num5 //= 3
print("Parte VI: valor da variavel num5 // 3 = %i " %num5)

d = 10
d /= 3 # d = d / 3
print (d)

d2 = 10//3
print (d2)

```

Figura B: exemplo31.py

Exemplo 32

PASTA: IPE-Python-2023/ Aula03-Python
arquivo: exemplo32.py

```

1  a, b, c = 2,4,8
2
3  print("O valor da variavel a = " +str(a))
4  print("O valor da variavel b = " + str(b))
5  print("O valor da variavel c = " + str(c))
6
7  a, b, c = a**2, b**3, c**4
8
9
10 print("Resultados da Potenciação e Radiciação")
11 print("O valor da Potencia de 2 da variavel a = " +str(a))
12 print("O valor da Potencia de 3 da variavel b = " + str(b))
13 print("O valor da Potencia de 4 da variavel c = " + str(c))
14 print("O valor da Potencia de 4 da variavel c = " + str(c))
15
16 print("Resultados Radiciação")
17 d = a**(1/2)
18 print("O valor da Radiciação de 2 da variavel 4 = " +str(d))
19 e = b**(1/3)
20 print("O valor da Radiciação de 2 da variavel 64 = " +str(e))
21 f = c**(1/4)

Resultados Radiciação
O valor da Radiciação de 2 da variavel 4 = 2.0
O valor da Radiciação de 2 da variavel 64 = 3.9999999999999996
O valor da Radiciação de 2 da variavel 4096 = 8.0
PS G:\IPE-Python2019>

```

Figura C: exemplo32.py

Exemplo 33

PASTA: IPE-Python-2023/ Aula03-Python
arquivo: exemplo33.py

Exemplo que determinar qual é estado (líquido, sólido ou vapor) da variável de entrada

```

1  num = int(input("Digite um número: "))
2
3  if(num > 0):
4  print("O número digitado = " + str(num) + " é maior do que 0")
5  else:
6  print("O número digitado = " + str(num) + " é menor do que 0")
7
8

```

```

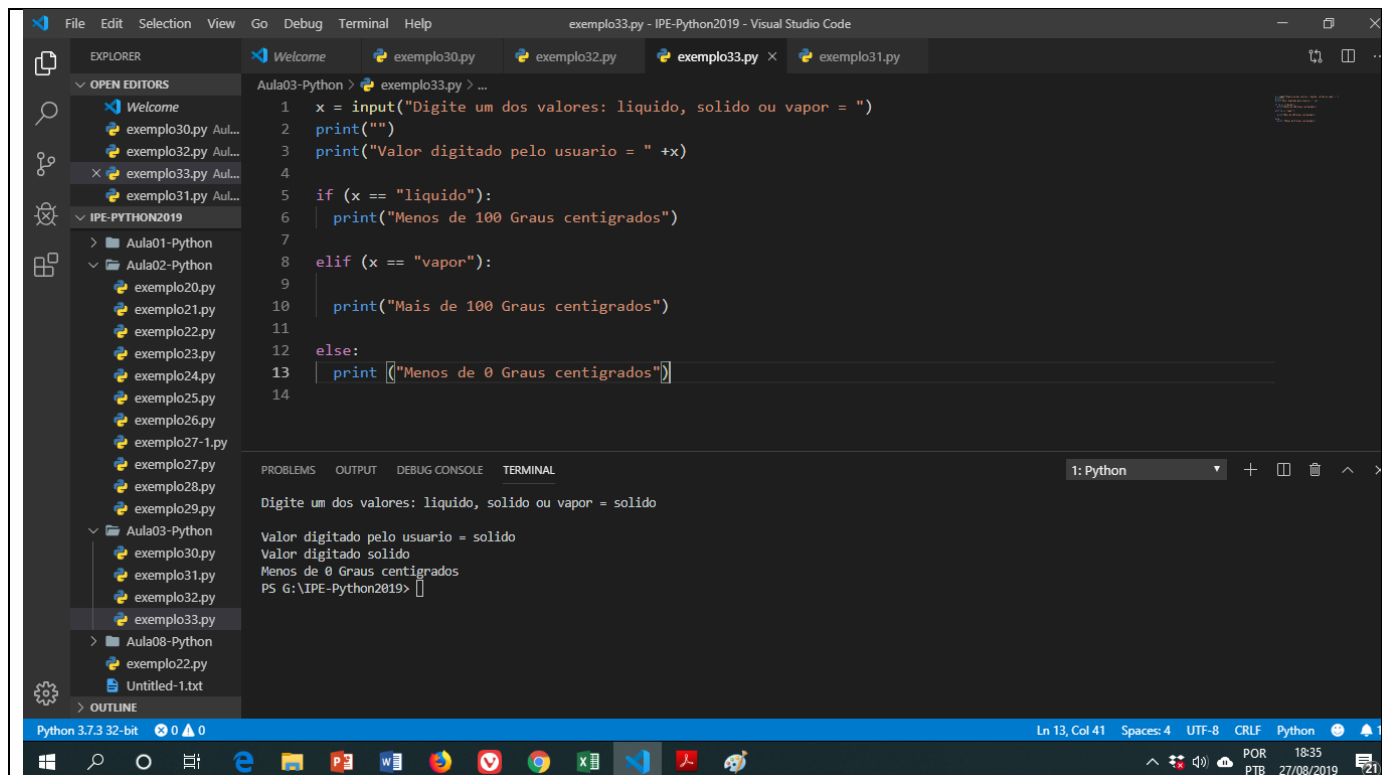
PS G:\IPE-Python2019> & C:\Users\SERGIOMEDINA\AppData\Local\Programs\Python\Python37-32\python.exe g:\IPE-Python2019/Aula03-Python/exemplo34.py
Digite um número: 7
O número digitado = 7 é maior do que 0
PS G:\IPE-Python2019>

```

Figura D: exemplo33.py

Exemplo 34

PASTA: IPE-Python-2023/ Aula03-Python
arquivo: exemplo34.py



```
1 x = input("Digite um dos valores: liquido, solido ou vapor = ")
2 print("")
3 print("Valor digitado pelo usuario = " +x)
4
5 if (x == "liquido"):
6     print("Menos de 100 Graus centigrados")
7
8 elif (x == "vapor"):
9     print("Mais de 100 Graus centigrados")
10
11 else:
12     print("Menos de 0 Graus centigrados")
13
14
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

1: Python

Digite um dos valores: liquido, solido ou vapor = solido

Valor digitado pelo usuario = solido

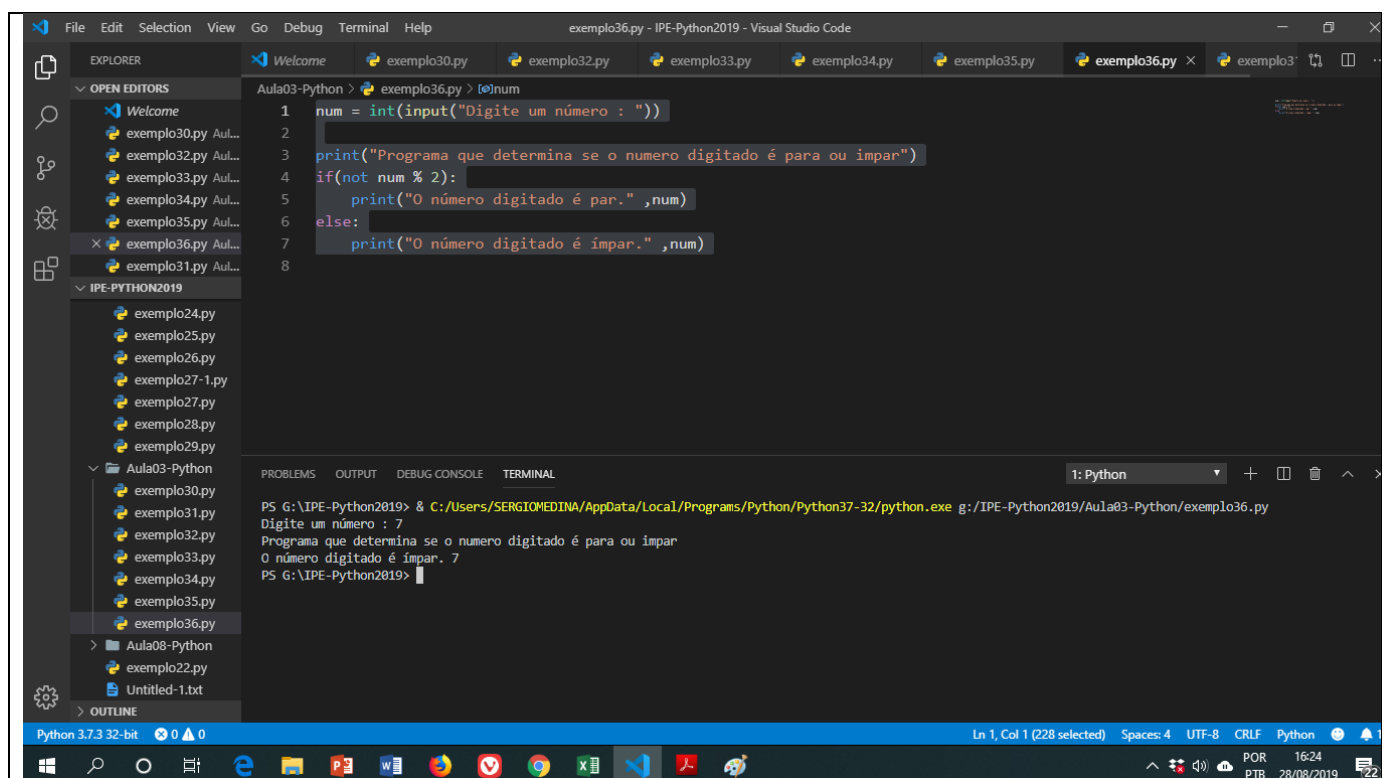
Menos de 0 Graus centigrados

PS G:\IPE-Python2019>

Figura E: exemplo34.py

Exemplo 35

PASTA: IPE-Python-2023/ Aula03-Python
arquivo: exemplo35.py



```
1 num = int(input("Digite um número : "))
2
3 print("Programa que determina se o numero digitado é para ou impar")
4 if(not num % 2):
5     print("O número digitado é par." , num)
6 else:
7     print("O número digitado é impar." , num)
8
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

1: Python

PS G:\IPE-Python2019> & C:\Users\SERGIOMEDINA\AppData\Local\Programs\Python\Python37-32\python.exe g:\IPE-Python2019/Aula03-Python/exemplo35.py

Digite um número : 7

Programa que determina se o numero digitado é para ou impar

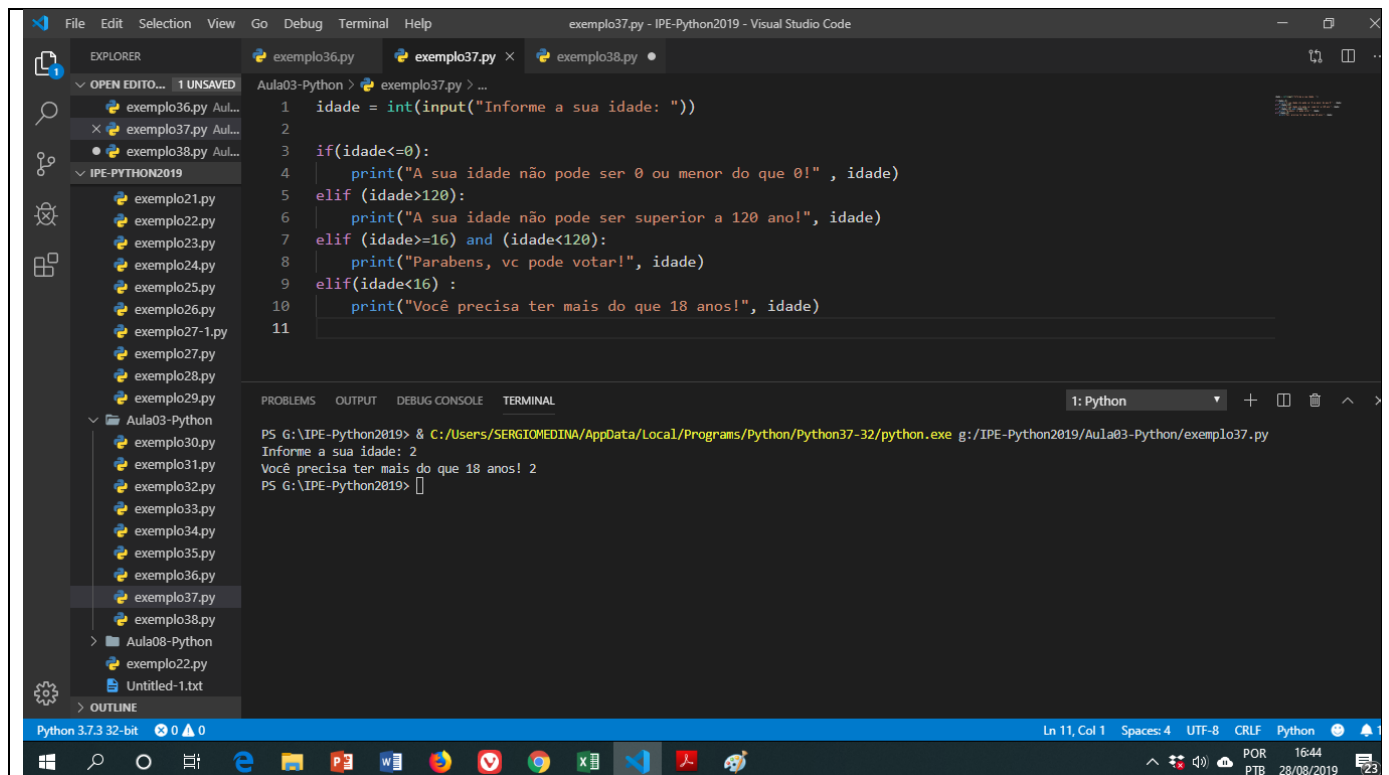
O número digitado é impar. 7

PS G:\IPE-Python2019>

Figura F: exemplo35.py

Exemplo 36

PASTA: IPE-Python-2023/ Aula03-Python
arquivo: exemplo36.py



The screenshot shows the Visual Studio Code interface with the file explorer on the left displaying a directory named 'IPE-PYTHON2019' containing various 'exemplo' files. The main editor window shows the code for 'exemplo36.py':

```
1 idade = int(input("Informe a sua idade: "))
2
3 if(idade<=0):
4     print("A sua idade não pode ser 0 ou menor do que 0!", idade)
5 elif (idade>120):
6     print("A sua idade não pode ser superior a 120 ano!", idade)
7 elif (idade>=16) and (idade<120):
8     print("Parabens, vc pode votar!", idade)
9 elif(idade<16) :
10    print("Você precisa ter mais do que 18 anos!", idade)
11
```

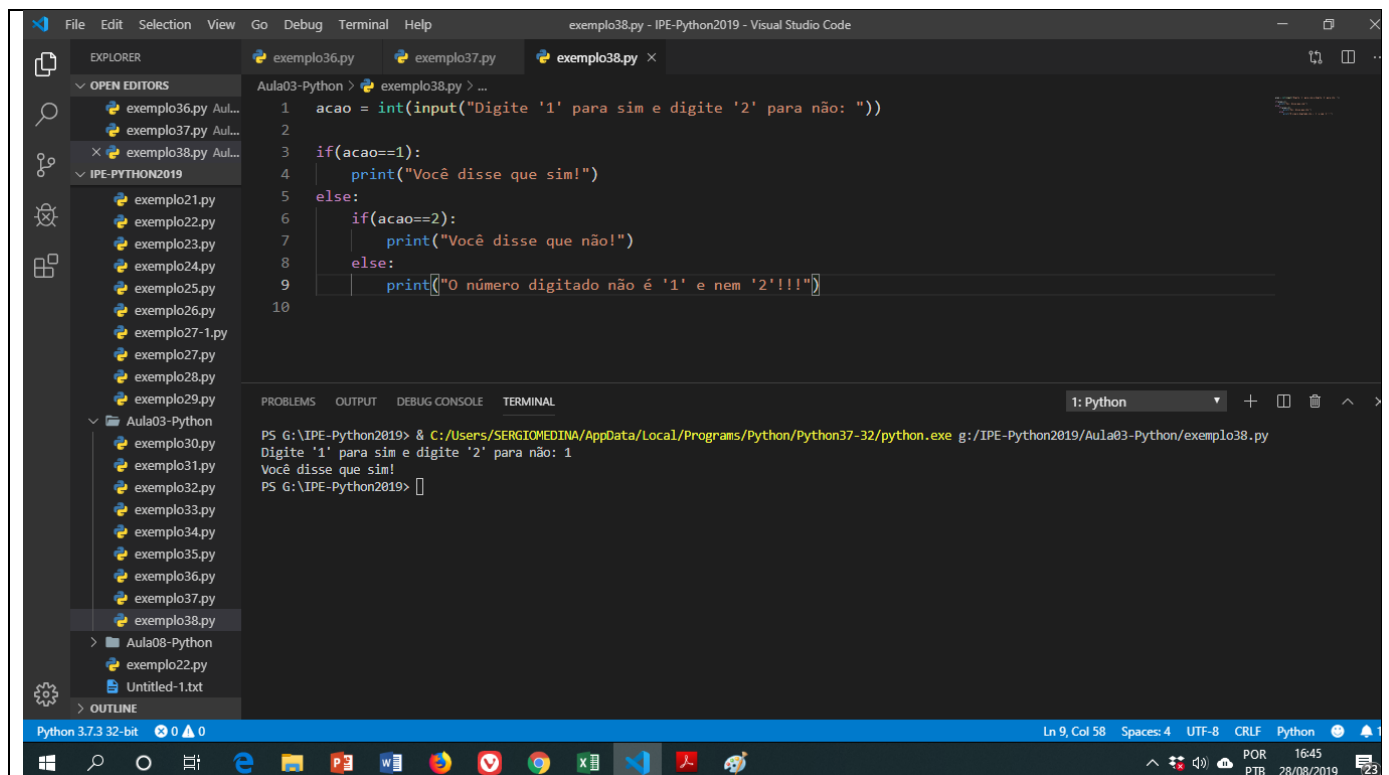
The terminal at the bottom shows the execution of the script:

```
PS G:\IPE-Python2019> & C:\Users\SERGIOMEDINA\AppData\Local\Programs\Python\Python37-32\python.exe g:/IPE-Python2019/Aula03-Python/exemplo37.py
Informe a sua idade: 2
Você precisa ter mais do que 18 anos! 2
PS G:\IPE-Python2019>
```

Figura G: exemplo36.py

Exemplo 37

PASTA: IPE-Python-2023/ Aula03-Python
arquivo: exemplo37.py



The screenshot shows the Visual Studio Code interface with the file explorer on the left displaying a directory named 'IPE-PYTHON2019' containing various 'exemplo' files. The main editor window shows the code for 'exemplo37.py':

```
1 acao = int(input("Digite '1' para sim e digite '2' para não: "))
2
3 if(acao==1):
4     print("Você disse que sim!")
5 else:
6     if(acao==2):
7         print("Você disse que não!")
8     else:
9         print("O número digitado não é '1' e nem '2'!!!")
10
```

The terminal at the bottom shows the execution of the script:

```
PS G:\IPE-Python2019> & C:\Users\SERGIOMEDINA\AppData\Local\Programs\Python\Python37-32\python.exe g:/IPE-Python2019/Aula03-Python/exemplo38.py
Digite '1' para sim e digite '2' para não: 1
Você disse que sim!
PS G:\IPE-Python2019>
```

Figura H: exemplo37.py

Exemplo 38

PASTA: IPE-Python-2023/ Aula03-Python
arquivo: exemplo38.py

```

1 print( "A relação (1 > 2) é ", 1 > 2 )
2 print( "a" > "b" )
3 print( "A relação (5 < 10) é ", 5 < 10 )
4 print( "A relação (200 == 200) é ", 200 == 200 )
5
6 print("A relação (1 >= 1) é : ", 1 >= 1 )
7 print("A relação (1 <= 1) é : ", 1 <= 1 )
8
9 print( "A relação (1 == 1) é ", 1 == 1 ) #essa expressão equivale a expressão seguinte
10 print( (1 == 1) == True )
11
12 print( (1 == 1) == 1 )
13 print( (1 == 1) == True )
14 print( (1 > 1) == 0 )
15 print( (1 > 1) == False )
16
17 print( 1 == True )
18 print( 0 == False )

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```

A relação (1 > 2) é False
False
A relação (5 < 10) é True
A relação (200 == 200) é True
A relação (1 >= 1) é : True
A relação (1 <= 1) é : True
A relação (1 == 1) é True
True
True
True

```

Figura I: exemplo38.py

Exemplo 39

PASTA: IPE-Python-2023/ Aula03-Python
arquivo: exemplo39.py

```

print("Programa que converte dias das semana numerico em texto")
numero_dia = int(input("Informe um numero para saber o dia da semana (1, 2, 3, 4, 5, 6, 7): "))
texto = "Um bom dia para aprender Python"
if (numero_dia == 1):
    print("Domingo. " + texto)
elif (numero_dia == 2):
    print ("Segunda feira. " + texto)
elif (numero_dia == 3):
    print ("Terca feira. " + texto)
elif (numero_dia == 4):
    print ("Quarta feira. " + texto)
elif (numero_dia == 5):
    print ("Quinta feira. " + texto)
elif (numero_dia == 6):
    print ("Sexta feira. " + texto)
elif (numero_dia == 7):
    print ("Sabado. " + texto)
else:
    print ("Valor invalido. Favor digitar numeros entre o intervalo (1, 2, 3, 4, 5, 6, 7)")

print("Fim do programa")

```

Figura J: exemplo39.py

Exemplo 39a**PASTA:** IPE-Python-2023/ Aula03-Python**arquivo:** exemplo39a.py

```
dia = int(input("Digite um numero inteiro entre '1 e 7': "))
mensagem = "Hoje é um bom dia para aprender Python"

match dia:
    case 1:
        print(" Domingo. " + mensagem)
    case 2:
        print(" 2da Feira. " + mensagem)
    case 3:
        print(" 3ça Feira. " + mensagem)
    case 4:
        print(" 4ta Feira. " + mensagem)
    case 5:
        print(" 5ta Feira. " + mensagem)
    case 6:
        print(" 6ta Feira. " + mensagem)
    case 7:
        print(" Sabado. " + mensagem)
    case _:
        print(" Dia invalido")
```

Figura k: exemplo39a.py