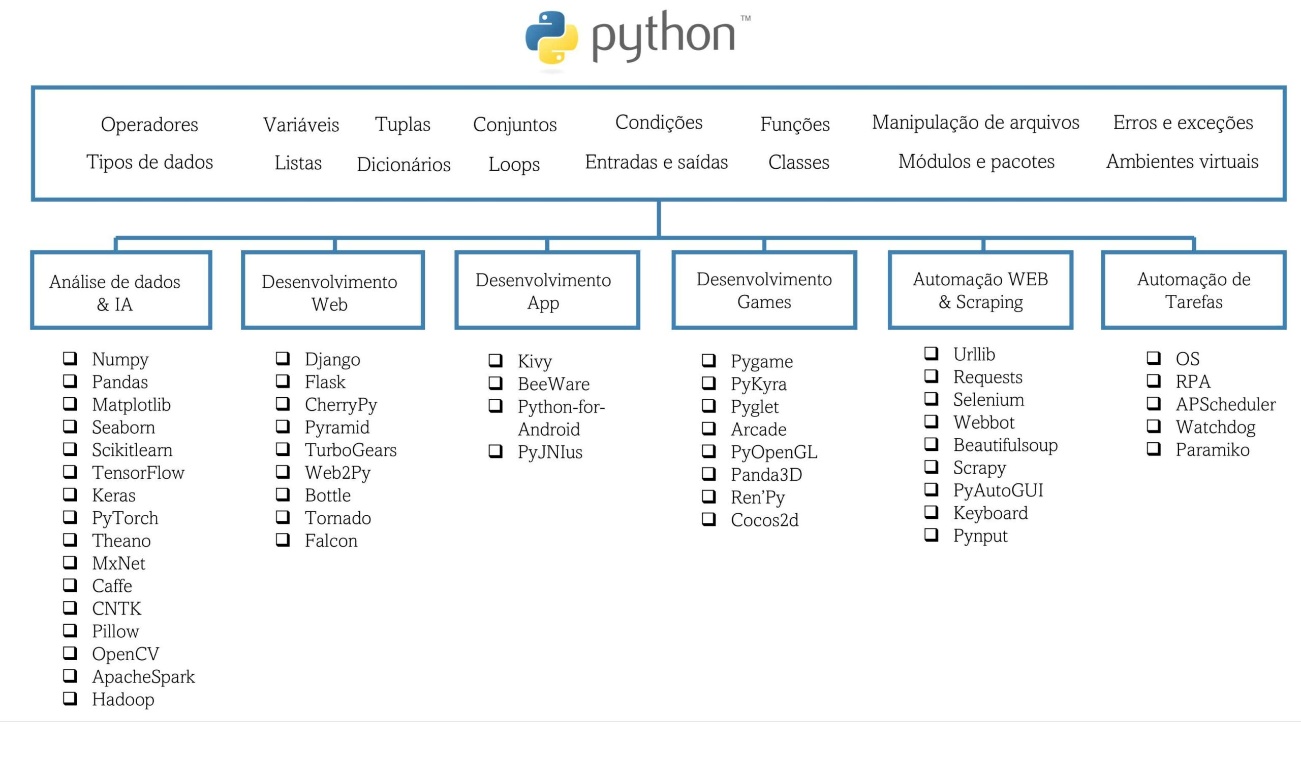
**Operadores aritmético**

Na linguagem Python temos a seguinte separação entre os diferentes tipos de operadores:

* OPERADORES ARITMÉTICOS
* OPERADORES DE ATRIBUIÇÃO
* OPERADORES DE COMPARAÇÃO
* OPERADORES LÓGICOS
* OPERADORES DE IDENTIDADE
* OPERADORES DE ASSOCIAÇÃO
* **Operadores aritméticos**
* Os operadores aritméticos são utilizados na execução de operações matemáticas, tais como a soma e a subtração, por exemplo. Vejamos tabela abaixo.

| **Operador** | **Conceito** | **Exemplo** |
| --- | --- | --- |
| + (Adição ou sinal positivo) | - Realiza a soma entre operandos - Adiciona o sinal de positivo ao número | - 10 + 7 - +4 |
| - (Subtração ou sinal negativo) | - Realiza a subtração entre operandos - Adiciona o sinal de negativo ao número | - 10 - 7 - -4 |
| \* (Multiplicação) | Realiza a multiplicação entre operandos | 3 \* 4 |
| / (Divisão) | Realiza a divisão entre operandos | 10 / 5 |
| // (Divisão inteira) | Realiza a divisão entre operandos e a parte decimal do resultado | 10 // 6 |
| % (Módulo) | Retorna o resto da divisão entre operandos | 4 % 2 |
| \*\* (Exponenciação) | Retorna um número elevado a potência de outro | 4 \*\* 2 |

**Exemplos:**



Uma característica importante a ser observada quando falamos dos operadores matemáticos é a precedência (preferência, ou seja, quem será calculado primeiro). Essa característica é relativa à ordem da execução deles e acontece quando mais de um operador está presente numa expressão. Segue a precedência dos operadores no Python.

Nos exemplos abaixo usaremos a função print() para exibir os resultados das expressões.

1. As expressões contidas em parênteses têm a precedência maior na linguagem Python. Isso permite que uma expressão execute antes de outra. Ex.:



1. Após os parênteses, o próximo operador com maior precedência é o de exponenciação. Ex.:



1. Multiplicação e divisão têm precedência sobre a adição e subtração: como já é conhecido na matemática, divisão e multiplicação são executadas antes das operações de adição e subtração. Ex.:



1. Ordem de precedência é avaliada da esquerda para a direita. Portanto, após os operadores anteriores, a sequência da execução será da esquerda para a direita. Ex.:

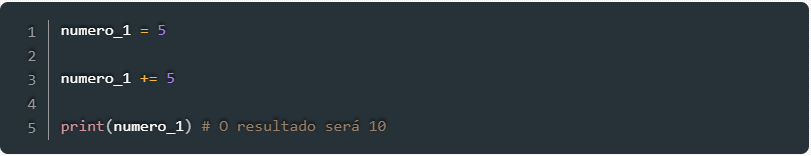


**Operadores de atribuição**

Os operadores de atribuição atribuem valor a uma variável. Na **Tabela 2** temos uma lista desses operadores.

| **Operador** | **Exemplo** | **Equivalente a** |
| --- | --- | --- |
| = | x = 1 | x = 1 |
| += | x += 1 | x = x + 1 |
| -= | x -= 1 | x = x - 1 |
| \*= | x \*= 1 | x = x \* 1 |
| /= | x /= 1 | x = x / 1 |
| %= | x %= 1 | x = x % 1 |

No **Código 2** temos um exemplo do uso de operadores de atribuição.



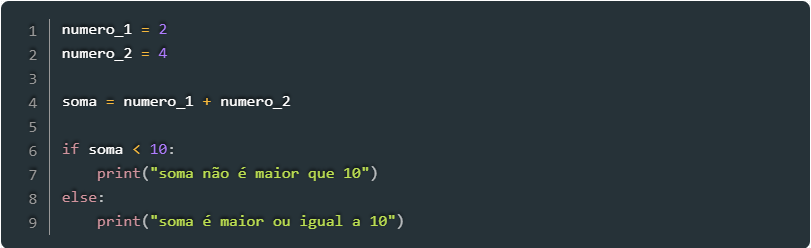
Acima, vemos que a variável numero\_1 recebe na linha 3 o valor dela mesmo somado a 5. Isso fará com que o resultado impresso na linha 5 seja igual a 10.

### Operadores de comparação

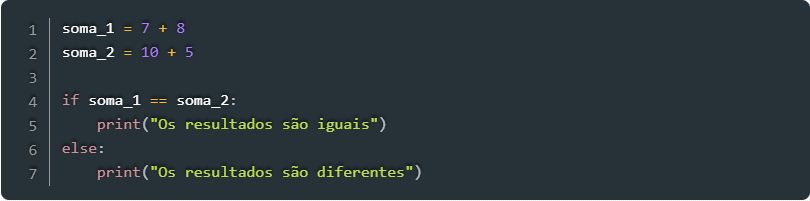
Os operadores de comparação são usados para comparar valores, o que vai retornar True ou False, dependendo da condição. A seguir, na **Tabela 3** temos exemplos de alguns usados no Python.

| **Operador** | **Conceito** | **Exemplo** |
| --- | --- | --- |
| >(Maior que) | Verifica se um valor é maior que outro | x > 5 |
| <(Menor que) | Verifica se um valor é menor que outro | x < 5 |
| == (Igual a) | Verifica se um valor é igual a outro | x == 5 |
| != (Diferente de) | Verifica se um valor é diferente de outro | x != 5 |
| >= (Maior ou igual a) | Verifica se um valor é maior ou igual a outro | x >= 5 |
| <= (Menor ou igual a) | Verifica se um valor é menor ou igual a outro | x <= 5 |

Usando a comparação que é feita com esses operadores, podemos criar condições para os códigos. No exemplo abaixo, vemos como isso acontece na prática:



Como podemos ver no **Código 3**, duas variáveis são somadas e esse resultado é atribuído à variável soma. Na linha 6, fazemos uma comparação dessa variável verificando se ela é menor do que 10. Vejamos a seguir, uma operação que faz comparação de igualdade entre duas variáveis:

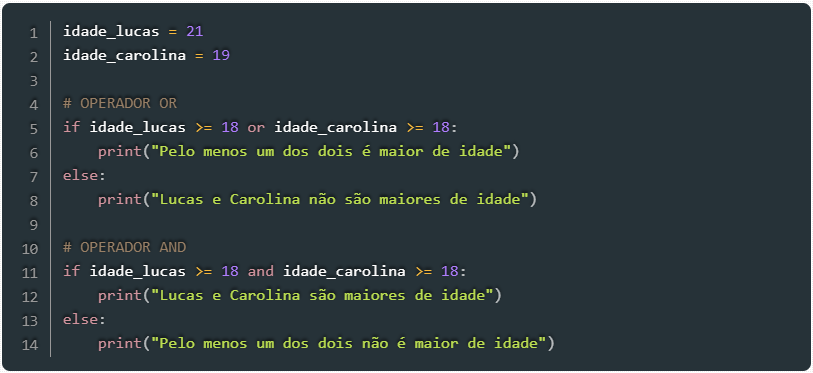


No **Código 4**, criamos duas variáveis e comparamos seus valores, o que pode resultar em dois resultados distintos. Dessa forma, podemos criar estruturas condicionais com o auxílio desses operadores.

**Operadores lógicos**

Os operadores lógicos são usados para unir duas ou mais expressões condicionais. Isso é feito por meio de conectivos, como podemos ver na **Tabela 4**.

|  |  |  |  |
| --- | --- | --- | --- |
| **Operador** | **Nome** | **Descrição** | **Tipo** |
| **and** | **Conjunção** | Retorna True se ambos os operandos tiverem o valor True | Binário |
| **or** | **Disjunção** | Retorna True se um dos operandos tiver o valor True | Binário |
| **not** | **Negação** | Inverte o resultado lógico | Unário |

Vejamos a seguir, um exemplo que utiliza os operadores lógicos:

No **Código 5** temos um exemplo no qual usamos os operadores or e and. Com o operador or, apenas uma das condições precisa ser verificadas. Já no operador and, todas as condições precisam ser satisfeitas. Com a execução do código teremos a impressão das mensages “Lucas e Carolina são maiores de idade” e "Pelo menos um dos dois é maior de idade".

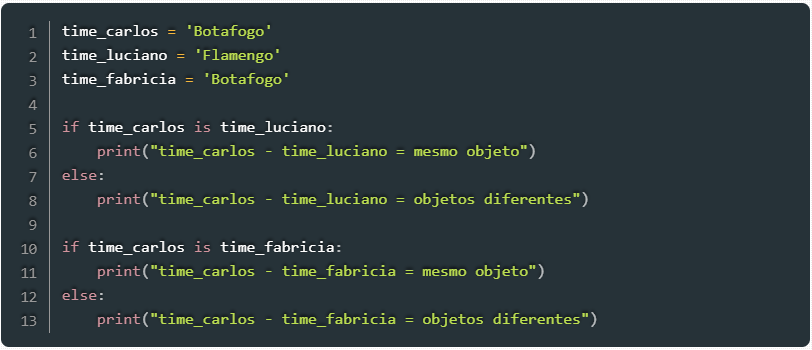
**Operadores de identidade**

Os operadores de identidade servem para a comparação de objetos. Nessa comparação é verificado se eles ocupam a mesma posição na memória, o que significará que se trata do mesmo objeto, como vemos isso na **Tabela 5**.

| **Operador** | **Conceito** | **Exemplo** |
| --- | --- | --- |
| is | Retorna True se as variáveis comparadas forem o mesmo objeto | nome is ‘Marcos’ |
| is not | Retorna True se as variáveis comparadas não forem o mesmo objeto | x is not ‘Python’ |

**Tabela 5**. Operadores de identidade

Vejamos com mais detalhes o uso dos operadores de identidade:



No **Código 6** temos um exemplo do operador is no Python. Com ele, o que está sendo verificado é se as variáveis apontam para o mesmo objeto. Quando declaramos uma string no Python, o seu valor é guardado num local especial da memória. Assim, quando usamos esse valor repetidas vezes como nas linhas 1 e 3, o Python pode utilizar o mesmo objeto em cada uma delas. Por esse motivo is dirá que time\_carlos e time\_fabricia apontam para o mesmo objeto, enquanto que time\_luciano aponta para um objeto diferente.

[**Neste artigo**](https://www.devmedia.com.br/tipos-de-dados-em-python-string/40669), podemos ver strings em mais detalhes.

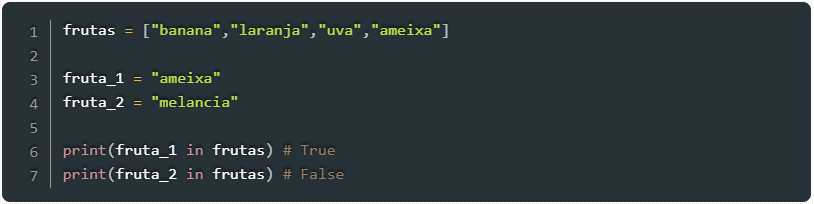
**Operadores de associação**

Os operadores de associação são utilizados para verificar se uma sequência contém um objeto. Na **Tabela 6** temos os exemplos.

| **Operador** | **Conceito** | **Exemplo** | |
| --- | --- | --- | --- |
|  |  |  |
| in | Retorna True caso o valor seja encontrado na sequência | 2 in x | |
| not in | Retorna True caso o valor não seja encontrado na sequência | 2 not in x | |

**Tabela 6**. Operadores de associação

No **Código 7**, vamos ver um exemplo usando os operadores de associação.



O exemplo acima verifica a existência de variáveis dentro de uma lista. Na linha 6 estamos checando se fruta\_1 está contido em frutas. Da mesma forma, fazemos isso na linha 7, só que dessa vez com fruta\_2.

### Conclusão

Ao usar os operadores no Python, podemos atribuir valores a variáveis, criar condições, fazer comparações etc. Essas características fazem parte dos princípios básicos para montar algoritmos nessa linguagem. É importante dominar esses conceitos sobre os operadores, pois eles são amplamente usados no dia a dia do desenvolvedor.

**Tipos de dados**

Python é uma linguagem dinamicamente tipada, o que significa que não é necessário declarar o tipo de variável ou fazer casting (mudar o tipo de variável), pois o Interpretador se encarrega disso para nós!

Isso significa também que o tipo da variável poder variar durante a execução do programa.

**Os tipos de dados padrão do Python são:**

* Inteiro (int)
* Ponto Flutuante ou Decimal (float)
* Tipo Complexo (complex)
* String (str)
* Boolean (bool)
* List (list)
* Tuple
* Dictionary (dic)

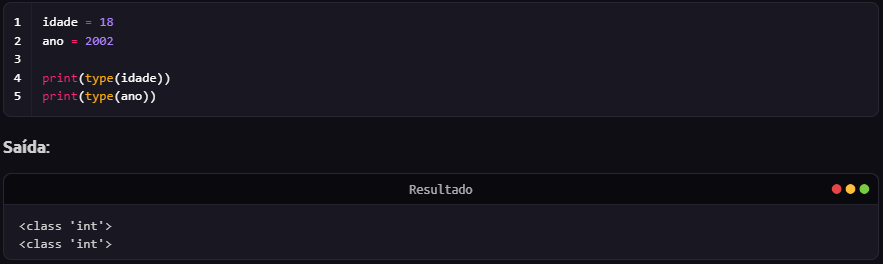
**Tipo Inteiro (int)**

O tipo inteiro é um tipo composto por caracteres numéricos (algarismos) inteiros.

É um tipo usado para um número que pode ser escrito sem um componente decimal, podendo ter ou não sinal, isto é: ser positivo ou negativo.

Por exemplo, 21, 4, 0, e −2048 são números inteiros, enquanto 9.75, 1/2, 1.5 não são.

Exemplos:

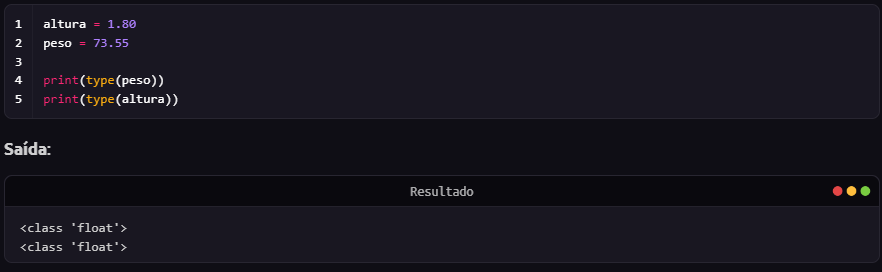


**Ponto Flutuante ou Decimal (float)**

É um tipo composto por caracteres numéricos (algarismo) decimais.

O famoso ponto flutuante é um tipo usado para números racionais (números que podem ser representados por uma fração) informalmente conhecido como “número quebrado”.

Exemplos:



**Complexo (complex)**

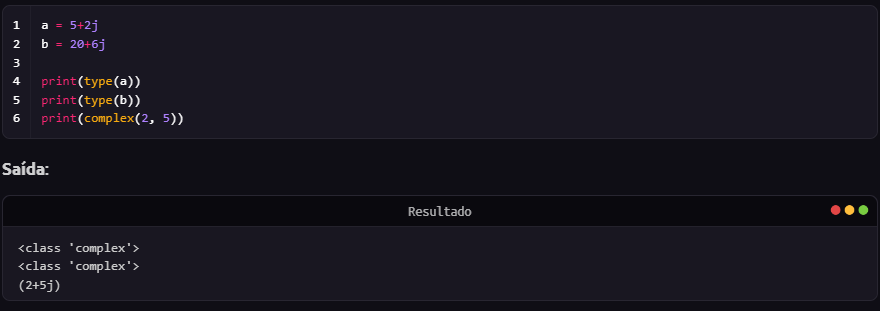
Tipo de dado usado para representar números complexos (isso mesmo, aquilo que provavelmente estudou no terceiro ano do ensino médio).

Esse tipo normalmente é usado em cálculos geométricos e científicos.

Um tipo complexo contem duas partes: a parte real e a parte imaginária, sendo que a parte imaginária contem um j no sufixo.

A função complex(real[, imag]) do Python possibilita a criação de números imaginários passando como argumento: real, que é a parte Real do número complexo e o argumento opcional imag, representando a parte imaginária do número complexo.

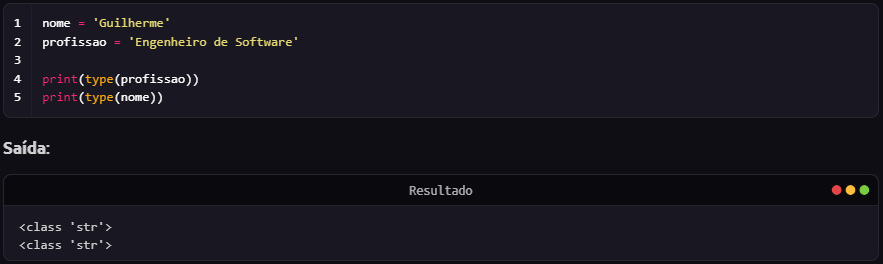
Exemplos:



**String (str)**

É um conjunto de caracteres dispostos numa determinada ordem, geralmente utilizada para representar palavras, frases ou textos.

Exemplos:

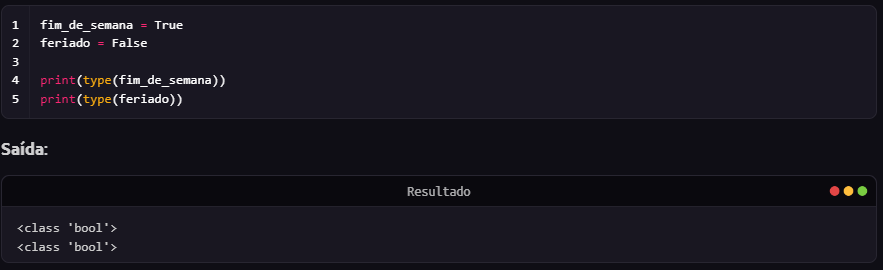


**Boolean (bool)**

Tipo de dado lógico que pode assumir apenas dois valores: falso ou verdadeiro (False ou True em Python).

Na lógica computacional, podem ser considerados como 0 ou 1.

Exemplos:

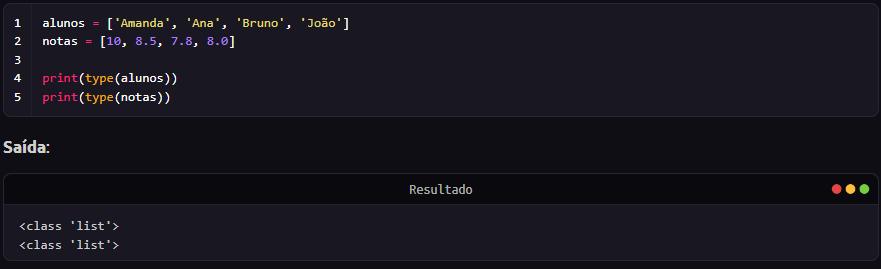


**Listas (list)**

Tipo de dado muito importante e que é muito utilizado no dia a dia do desenvolvedor Python!

Listas agrupam um conjunto de elementos variados, podendo conter: inteiros, floats, strings, outras listas e outros tipos.

Elas são definidas utilizando-se colchetes para delimitar a lista e vírgulas para separar os elementos, veja alguns exemplo abaixo:



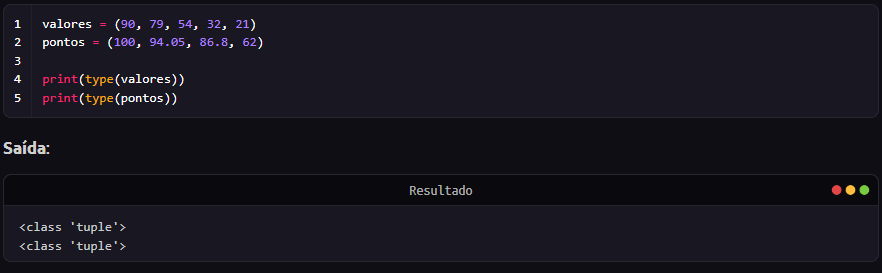
**Tuplas (tuple)**

Assim como Lista, Tupla é um tipo que agrupa um conjunto de elementos.

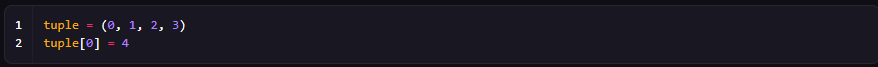
Porém sua forma de definição é diferente: utilizamos parênteses e também separado por vírgula.

A diferença para Lista é que Tuplas são imutáveis, ou seja, após sua definição, Tuplas não podem ser modificadas.

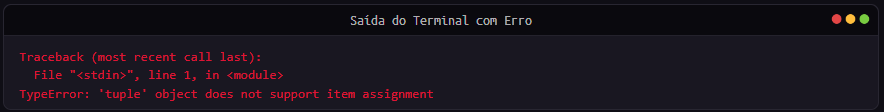
Vamos ver alguns exemplos:



Caso haja uma tentativa de alterar os itens de uma tupla após sua definição, como no código a seguir:



O seguinte erro do tipo TypeError será lançado pelo interpretador do Python:

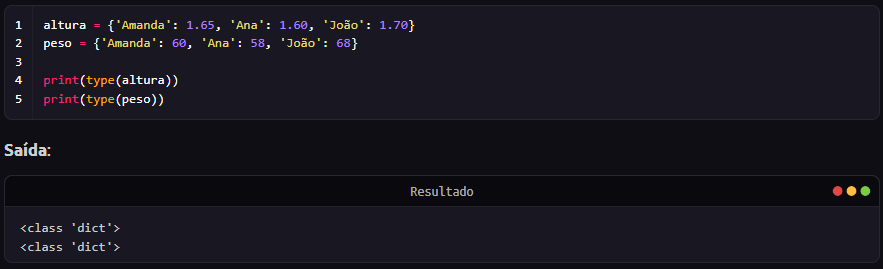


**Dicionários (dict)**

Dict é um tipo de dado muito flexível do Python.

Eles são utilizados para agrupar elementos através da estrutura de chave e valor, onde a chave é o primeiro elemento seguido por dois pontos e pelo valor.

Vamos ver alguns exemplos:

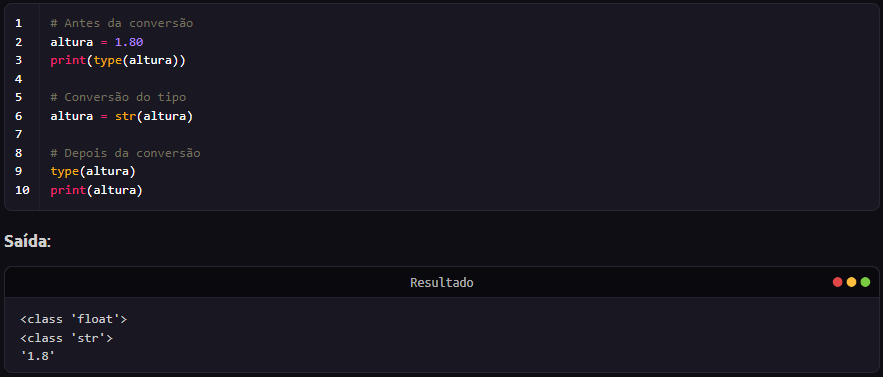


**Como mudar o tipo de uma variável**

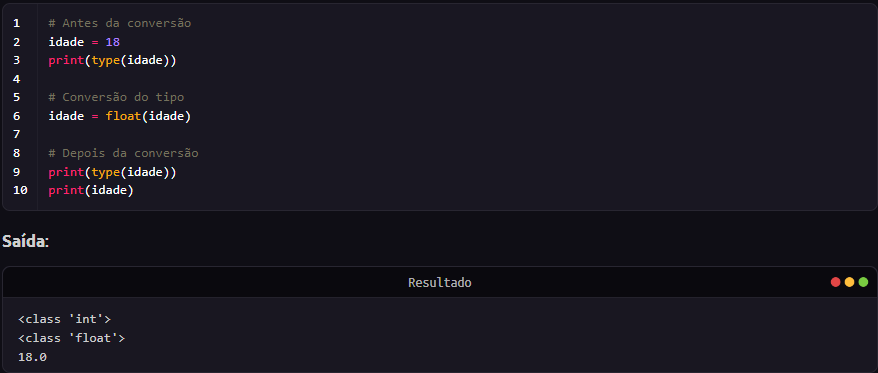
Em determinados cenários pode ser necessário mudar o tipo de uma variável e no Python isso é muito fácil, uma das vantagens de uma linguagem dinamicamente tipada.

Abaixo veremos exemplos de como trocar o tipo de variáveis.

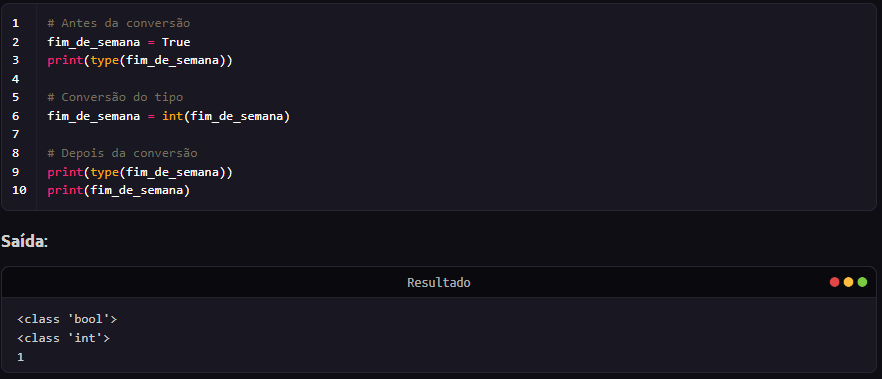
**Decimal (float) para String (str)**



**Inteiro (int) para Decimal (float):**



**Booleano (bool) para Inteiro (int):**



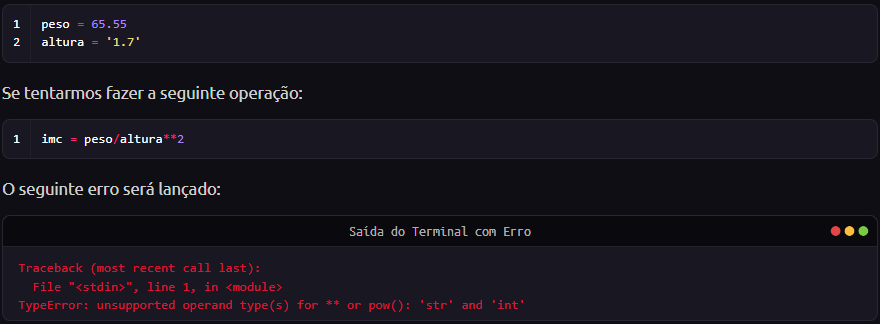
**Erros comuns relacionados ao tipo da variável**

Entre os erros mais comuns que acontecem principalmente com quem está iniciando sua jornada com a linguagem Python estão os famosos TypeError!

Portanto preste sempre atenção nesse tipo de erro pois ele está relacionado a tipagem errada de uma variável ou objeto.

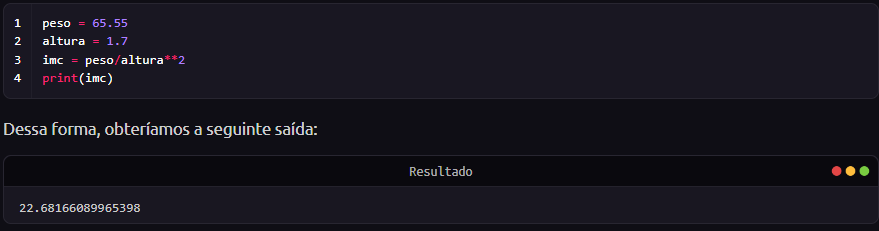
Além disso, pode representar uma operação incompatível, como tentar fazer uma operação matemática com variáveis de tipo incompatíveis.

Por exemplo, suponha as seguintes variáveis, do tipo inteiro e string, respectivamente:



Para fazer esse cálculo corretamente a altura não pode ser string, podendo ser float ou inteiro (caso não fosse um valor decimal).

Uma forma correta de realizar esse cálculo seria da seguinte maneira:



Sempre que o tipo TypeError aparecer em seu programa, preste atenção aos tipos dos dados e as operações que estão sendo executadas.

Caso seja necessário, faça a conversão das variáveis para se trabalhar com os tipos correto!