Variáveis, entradas e saídas

1. Variáveis

Em nossos programas, frequentemente precisaremos armazenar dados temporariamente. Esses dados podem ser adquiridos de alguma maneira (digitados pelo teclado, lidos de um arquivo etc.) ou calculados pelo nosso programa com base em outros dados. Imagine, por exemplo, que você gostaria de calcular a média de um aluno a partir de suas notas. Precisaremos que o aluno digite suas notas, e o programa irá calcular um novo valor, a média. Armazenaremos nossos dados temporariamente em **variáveis**.

Variáveis são "pedacinhos de memória" onde guardamos dados. Sempre que referenciamos o nome, o pedacinho de memória é acessado e seu dado é recuperado.

Criamos variáveis dando um nome a elas e usando o operador de atribuição (o sinal de igualdade: =) para atribuir um valor inicial.

```
x = 10
```

No exemplo acima, foi criada uma variável chamada **x** que guarda o valor **10**. Ou seja, reservamos um pedacinho de memória e guardamos o número 10 lá.

Tente sempre utilizar nomes intuitivos para suas variáveis. O nome deveria ser uma boa descrição do dado que a variável guarda. Nomes como 'x', 'y', 'z', 'a', 'b', 'c', 'a1', 'a2', 'a3' etc. podem se tornar bastante confusos quando nossos códigos são muito grandes. Quanto mais descritivos os nomes forem, melhor.

Os nomes de variáveis podem conter letras, números e o símbolo `_`, mas eles não podem começar com número.

> Dica: existe uma grande variedade de padrões diferentes que podemos adotar para nomear nossas variáveis. Em Python é recomendável utilizar o padrão conhecido como *snake case*, em que nomes de variáveis com múltiplas palavras adotam o símbolo `_` para separar as palavras. Exemplos: `nome_completo`, `nota_da_prova` etc.

2. Tipos de variáveis

Variáveis podem ter diferentes tipos. Alguns tipos são considerados **tipos primitivos**, ou seja, eles são tipos de dados mais básicos que podem ser utilizados para compor outros tipos mais complexos. Em Python esses tipos levam os seguintes nomes:

int: números *inteiros*, ou seja, números sem parte decimal: 0, 5, -1, 1000

float: números *reais*, ou seja, números com parte decimal: 1.0, -2.7, 3.14

str: cadeias de *caracteres* (*strings*), ou seja, dados textuais: 'Olá Mundo!', "eu tenho 18 anos"

bool: valores *lógicos* (*booleanos*), ou seja, apenas um entre dois valores possíveis: *True* ou *False*

```
nome = 'Zé' # uma variável do tipo string - note as aspas
email = "ze@letscode.com.br" # outra string
idade = 22 # uma variável inteira
salario = 5999.85 # uma variável float - usamos ponto, não vírgula
receber_newsletter = True # uma variável bool
```

O Python é uma linguagem **dinamicamente tipada**. Isso significa que não precisamos especificar o tipo de uma variável: a própria linguagem tenta determinar o tipo de acordo com o dado atribuído à variável.

3. Comentários

Note que nos exemplos acima, escrevemos textos no meio do código utilizando o símbolo **#**. Esses textos são **comentários**: quando utilizamos o símbolo **#**, o Python irá ignorar tudo o que vier em seguida (na mesma linha). Utilizamos comentários para explicar pedaços do nosso código para que nós mesmos ou outros colegas no futuro entendam o que fizemos e possam modificar ou corrigir o código com mais facilidade. Também podemos escrever comentários de múltiplas linhas utilizando aspas triplas - neste caso, as utilizamos para abrir e depois para fechar o bloco de comentários.

```
'''
Este é um comentário de várias linhas.
Tudo que veio após o primeiro trio de aspas e antes do segundo
será ignorado pelo Python.
'''
```

Na verdade, esse tipo de comentário não é exatamente um comentário, mas uma string com múltiplas linhas. O Python enxerga que apenas "declaramos" uma string no meio do código, sem utilizá-la ou atribuí-la para qualquer variável, e por conta disso ela é ignorada, funcionando na prática como um comentário.

> Na maioria das IDEs você possui teclas de atalho para facilmente transformar um bloco inteiro de código em comentário para temporariamente desabilitá-lo. Isso pode ser útil quando estamos testando soluções alternativas para um problema ou corrigindo erros. No Visual Studio Code, por exemplo, você pode utilizar `ctrl+/` para transformar uma seleção em comentário.

4. Saídas

Chamamos de **saídas** do nosso programa todos os dados que são gerados pelo programa e serão fornecidos para o usuário. A função de saída em tela no Python é o **print**. Colocamos entre parênteses o dado que queremos que apareça.

```
print('olá mundo!') # exibe a frase 'olá mundo' na tela
```

Os dados a serem exibidos não precisam ser valores constantes, como a frase fixa acima. Eles podem ser variáveis:

```
idade = 20
print(idade)
```

> Note que quando usamos aspas, o Python trata o valor como uma *string*, um texto literal. Quando não usamos aspas, o Python irá considerar que aquele é o nome de uma variável e irá acessá-la para buscar seu valor.

Podemos exibir múltiplos dados em um **print**. Para isso, basta separá-los por vírgula e eles irão aparecer na tela na mesma ordem que apareceram no código:

```
nome = 'Mario'
linguagem = 'Python'
print('Oi, eu sou o', nome, 'e eu programo em', linguagem)
```

Resultado na tela:

```
Oi, eu sou o Mario e eu programo em Python
```

Note que os dados aparecem em tela separados por um espaço automaticamente. Dois prints sucessivos também possuem uma quebra de linha entre eles. Você pode passar as opções `sep` e `end` dentro de seu print para especificar diferentes comportamentos. Exemplo:

```
nome = 'Mario'
linguagem = 'Python'
print('Oi, eu sou o', nome, sep='@', end='***')
print('Eu programo em', linguagem, sep='@')
```

Resultado na tela:

```
Oi, eu sou o@Mario***Eu programo em@Python
```

> Dica: caso você ache confuso separar os dados por vírgulas, você pode alternativamente utilizar uma *f-string*. Não entraremos em detalhes agora, mas o funcionamento básico é simples: coloque um *f* antes de abrir aspas, e dentro do texto você pode colocar o nome das variáveis entre chaves. o `print` abaixo terá o mesmo resultado que o exemplo anterior:

```
print(f'Oi, eu sou o {nome} e eu programo em {linguagem}')
```

5. Entradas

Assim como temos dados de saída - dados gerados pelo código e fornecidos para o usuário - também temos dados de **entrada**: informações que o usuário possui e deve fornecer ao código. Para receber entradas pelo teclado, utilizaremos a função **input**. Devemos levar uma variável a receber o valor capturado pelo input.

```
nome = input()
print('Olá', nome)
```

O programa acima captura o nome do usuário e em seguida mostra a mensagem "olá" seguida do nome do usuário. Note que o programa fica parado em uma tela em branco com um cursor piscando aguardando a digitação pelo usuário. Isso pode ser confuso para o usuário, que não sabe o que o programa está esperando. Por isso, dentro dos parênteses do input podemos colocar uma mensagem simples informando o que o programa gostaria que ele fizesse:

```
nome = input('Qual é o seu nome?')
print('Olá', nome)
```

5.1. Determinando o tipo da entrada

Vamos imaginar um programa que informa quantos anos falta para que uma criança atinja a maioridade. Podemos ler a idade da criança pelo teclado (*entrada*), subtrair a idade do número 18 (*processamento*) e exibir o resultado da conta na tela (*saída*). Considere a solução abaixo:

```
idade = input('Digite a sua idade: ')
resto = 18 - idade
print('Faltam', resto, 'anos.')
```

Se você copiar e executar o programa, ele dará erro na segunda linha. Isso ocorre porque o teclado é uma "máquina de escrever" um pouco mais moderna. Portanto, tudo que entra pelo teclado é considerado pelo Python como texto (ou seja, *str*). Porém, não podemos "fazer contas" com textos. Fazemos contas com números. Portanto, neste caso, precisamos falar para o Python interpretar a nossa entrada como um número. Um bom tipo de dado para "idade" seria um número inteiro. Fazemos isso colocando o nome do tipo desejado, e entre parênteses colocamos nosso input:

```
idade = int(input('Digite a sua idade: '))
resto = 18 - idade
print('Faltam', resto, 'anos.')
```

Chamamos essa operação de **coerção de tipo**. Em materiais em inglês você verá essa operação com o nome *casting*. Tome cuidado: operações de coerção podem resultar em perdas de dados. Se você converter o número float 3.9 para int, ele **não** arredondará para 4, e sim descartará a parte fracionária, resultando em 3.

Neste início, mensagens de erro podem parecer intimidadoras. Elas aparecem em vermelho e frequentemente possuem nomes técnicos e expressões em inglês. Mas crie o hábito de tentar compreendê-las. A partir da versão 3.10 do Python elas se tornaram significativamente mais amigáveis. Elas também indicam a linha com erro. Além disso, se você pesquisar em sites de busca por uma mensagem de erro, provavelmente encontrará diversos exemplos e explicações do que pode tê-la provocado e como consertar!

6. Expressões aritméticas

Como podemos observar no exemplo anterior, o Python faz operações aritméticas de maneira bastante intuitiva, similar ao que estamos acostumados. Os operadores aceitos são:

Soma: +
Subtração: -
Multiplicação: *
Divisão: /
Divisão inteira: //
Resto da divisão: %
Potência: **

```
numero1 = int(input('Digite um número: '))
numero2 = int(input('Digite outro número: ''))
soma = numero1 + numero2
subtracao = numero1 - numero2
multiplicacao = numero1 * numero2
divisao_real = numero1 / numero2
divisao_inteira = numero1 // numero2
resto = numero1 % numero2
elevado = numero1 ** numero2
print('Soma: ', soma)
print('Subtração: ', subtracao)
print('Multiplicação: ', multiplicacao)
print('Divisão: ', divisao_real)
print('Divisão inteira: ', divisao_inteira)
print('Resto da divisão: ', resto)
print('Potência: ', elevado)
```

Operadores de divisão: Note que temos 3 operadores de divisão. O que seria cada um deles? Vamos supor que numero1 seja 15 e numero2 seja 6.

```
15 |__ 6
```

Quantas vezes o número 6 cabe dentro do 15? Um bom primeiro "chute" é 2:

```
 15 |__ 6
    2
```

Podemos multiplicar 6 por 2, que dará 12. E então subtraímos esse valor de 15:

```
 15 |__ 6
-12    2
---
 03
```

Note que, considerando apenas números inteiros, não conseguimos mais prosseguir com a divisão. Neste caso, a **divisão inteira** (`numero1 // numero2`) dará 2. Já o **resto da divisão** (`numero1 % numero2`) dará 3.

Porém, considerando casas decimais é possível prosseguir com a divisão:

```
 15 |__ 6
-12    2.5
---
 03
  30
- 30
----
   0
```

Portanto, a **divisão** real (`numero1 / numero2`) dará 2.5.

> **Atenção:** números reais em Python usam ponto para separar as casas decimais, não vírgula:
>
> **Errado:** 2,5
> **Correto:** 2.5