Variáveis

Na aula anterior, imprimimos textos com a função `print()` [doc]. Para recapitular:

In [1]:
```
print("Estamos na segunda aula!")
```

```
Estamos na segunda aula!
```

Mas e se a informação que queremos imprimir for muito extensa? Teremos de digitar tudo de novo sempre que quisermos imprimir?

Nesses casos, usamos uma **variável**.

Uma variável é **um nome que se refere a um valor** e fica temporariamente salvo na memória do computador. Sempre que chamarmos a variável, o valor será evocado.

A atribuição é simples:

```
variavel = valor
```

Vamos pegar como exemplo o primeiro parágrafo de "Alice no País das Maravilhas":

> Alice estava começando a ficar muito cansada de estar sentada ao lado de sua irmã e não ter nada para fazer: uma vez ou duas ela dava uma olhadinha no livro que a irmã lia, mas não havia figuras ou diálogos nele e "para que serve um livro", pensou Alice, "sem figuras nem diálogos?"

In [2]:
```
alice = 'Alice estava começando a ficar muito cansada de estar sentada ao lado de
```

Este texto imenso agora está salvo com o "apelido" de `alice`. Sempre que quisermos usar o texto podemos chamá-lo pela variável:

In [3]:
```
print(alice)
```

```
Alice estava começando a ficar muito cansada de estar sentada ao lado de sua irmã
e não ter nada para fazer: uma vez ou duas ela dava uma olhadinha no livro que a
irmã lia, mas não havia figuras ou diálogos nele e "para que serve um livro", pen
sou Alice, "sem figuras nem diálogos?"
```

Quer dizer... *quase* sempre. A variável perde seu valor quando:

1. o script é encerrado;
2. o valor é alterado.

Sim, eu posso mudar o valor de uma variável já existente. Basta atribuir novo valor:

In [4]:
```
print("Valor anterior: ", alice)
alice = "Então ela pensava consigo mesma (tão bem quanto era possível naquele dia
print("Valor novo: ", alice)
```

```
Valor anterior:  Alice estava começando a ficar muito cansada de estar sentada ao
lado de sua irmã e não ter nada para fazer: uma vez ou duas ela dava uma olhadinh
a no livro que a irmã lia, mas não havia figuras ou diálogos nele e "para que ser
ve um livro", pensou Alice, "sem figuras nem diálogos?"
Valor novo:  Então ela pensava consigo mesma (tão bem quanto era possível naquele
```

```
dia quente que a deixava sonolenta e estúpida) se o prazer de fazer um colar de m
argaridas era mais forte do que o esforço de ter de levantar e colher as margarid
as, quando subitamente um Coelho Branco com olhos cor-de-rosa passou correndo per
to dela.
```

Há algumas regras para criar variáveis:

1. não pode começar com número ou símbolo, mas pode ter número e `underscore` (`_`) no meio ou no fim;
2. não pode conter acentos ou pontos;
3. maiúsculas e minúsculas são diferentes (a variável `nome` não é igual à variável `Nome`);
4. não pode ser palavra-chave de Python: `and` , `as` , `assert` , `break` , `class` , `continue` , `def` , `del` , `elif` , `else` , `except` , `exec` , `finally` , `for` , `from` , `global` , `if` , `import` , `in` , `is` , `lambda` , `nonlocal` , `not` , `or` , `pass` , `raise` , `return` , `try` , `while` , `with` , `yield` , `True` , `False` , `None` .

Agora vamos fazer um teste lógico:

```
variavel_1 = "Ana"
variavel_2 = "Maria"
variavel_1 = variavel_2
print(variavel_1)
```

Qual será o resultado?

Tipos de dados

Nesta aula e na anterior, houve vezes em que usei valores com aspas, e outras vezes em que usei sem aspas. Exemplos:

```
pi = 3.1416
linguagem = "Python"
```

Isso porque os valores são de tipos diferentes: o primeiro pertence ao tipo numérico `float` (com decimal), enquanto o segundo pertence ao tipo não-numérico `string` (texto). Em Python, cada um tem suas particularidades.

Nesta aula, vamos ver os tipos básicos e como trabalhar com eles:

Numéricos

- Integer (`int`)
  - números inteiros
  - não usa aspas
- Float (`float`)
  - números decimais; números com ponto flutuante; notação científica
  - não usa aspas
- Complex (`complex`)
  - números complexos, com √-1 em parte da equação
  - em Python, número imaginário representado pela letra `j`
  - não usa aspas

In [5]:
```
print(32)
print(type(32)) # a função type() mostra o tipo de dado
```

```
32
<class 'int'>
```

In [6]:
```
print(32.0)
print(type(32.0))
```

```
32.0
<class 'float'>
```

In [7]:
```
print(0.000000000000000000000000000001)
print(type(0.000000000000000000000000000001))
```

```
1e-30
<class 'float'>
```

In [8]:
```
from cmath import sqrt  # veremos módulos nas próximas aulas
print(sqrt(-1))
print(type(sqrt(-1)))
```

```
1j
<class 'complex'>
```

Não-numérico

- String (`str`)
  - texto; sequência de caracteres alfanuméricos; letra
  - aparecem entre aspas duplas (`"`) ou simples (`'`)

In [9]:
```
print("abcde")
print(type("abcde"))
```

```
abcde
<class 'str'>
```

In [10]:
```
print("a")
print(type("a"))
```

```
a
<class 'str'>
```

In [11]:
```
print(1234)
print(type(1234))

print('1234') # reparem que, ao usar aspas, converto int em str
print(type('1234'))
```

```
1234
<class 'int'>
1234
<class 'str'>
```

Notem que, nas linhas acima, eu ora usei aspas duplas, ora usei aspas simples. Os dois tipos de aspas valem, mas cuidado:

In [12]:
```
print("Estou usando 'aspas simples' dentro de aspas duplas")
```

```
Estou usando 'aspas simples' dentro de aspas duplas
```

In [13]:
```
print('Estou usando "aspas duplas" dentro de aspas simples')
```

```
Estou usando "aspas duplas" dentro de aspas simples
```

In [14]:
```
print('O que acontece se, dentro de 'aspas simples', eu usar aspas simples?')
```

```
  File "<ipython-input-14-bfb62173fc6b>", line 1
    print('O que acontece se, dentro de 'aspas simples', eu usar aspas simples?')
                                          ^
SyntaxError: invalid syntax
```

In [15]:
```
print("O mesmo acontece com "aspas duplas" dentro de aspas duplas?")
```

```
  File "<ipython-input-15-9d3468675ccf>", line 1
    print("O mesmo acontece com "aspas duplas" dentro de aspas duplas?")
                                ^
SyntaxError: invalid syntax
```

Nos dois últimos exemplos, eu tive erro de sintaxe. Isso ocorre porque a máquina entende que os dados se encerram no fechamento de aspas e não sabe o que fazer com o restante da informação:

- No primeiro exemplo, a máquina só computou `'O que acontece se, dentro de '`;
- No segundo exemplo, a máquina computou `"O mesmo acontece com "`.

Uma opção é usar `escape` (`\`) nas aspas internas. `Escape` funciona para eu avisar a máquina: "interprete de maneira literal, como aspas; não considere como fim de um texto".

In [16]:
```
print('E agora? Consigo usar \'aspas simples\' dentro de aspas simples?')
```

```
E agora? Consigo usar 'aspas simples' dentro de aspas simples?
```

In [17]:
```
print("E \"aspas duplas\" dentro de aspas duplas?")
```

```
E "aspas duplas" dentro de aspas duplas?
```

Lógico

- Boolean (`bool`)
  - comporta apenas dois valores: `True` (verdadeiro) e `False` (falso)
  - nos "bastidores", funciona como número, sendo `0` para `False` e `1` para `True`
  - assim como números, não usa aspas

In [18]:
```
print(True)
print(type(True))
```

```
True
<class 'bool'>
```

In [19]:
```
print(False)
print(type(False))
```

```
False
<class 'bool'>
```

In [20]:
```
print(2 > 3)
```

```
False
```

In [21]:
```
print(2 < 3)
```

```
True
```

Operações aritméticas

Agora que sabemos o que são **variáveis** e conhecemos os **tipos básicos**, podemos juntar as duas coisas para resolver **operações matemáticas**. Por exemplo, descobrir o salário mínimo por dia:

In [22]:
```
sal_minimo = 1100
dias_uteis = 22
print(sal_minimo / dias_uteis)
```

```
50.0
```

Ou, como no exercício da aula anterior, calcular a área de um círculo de 12,7 metros de raio:

In [23]:
```
raio = 12.7
pi = 3.14159
area = pi * (raio**2)
print(area)
```

```
506.70705109999994
```

No primeiro exemplo, eu dividi o salário mínimo (`sal_minimo`) pela quantidade de dias úteis (`dias_uteis`) em um mês. Para isso, usei o símbolo de divisão (`/`).

No segundo exemplo, elevei o raio (`raio`) ao quadrado usando o símbolo de exponenciação (`**`) e, a ele, multipliquei o valor de pi (`pi`) com o símbolo de multiplicação (`*`).

Os operadores são:

- adição: `+`
- subtração: `-`
- multiplicação: `*`
- divisão: `/`
- exponenciação: `**`
- parte inteira (descarta decimais): `//`
- módulo (o resto de uma divisão): `%`

Cabe ressaltar: a ordem de execução de operações segue a ordem convencional na matemática:

1. exponenciação
2. multiplicação e divisão
3. soma e subtração

Podemos sobrescrever essa ordem usando parênteses.

In [24]:
```
print(2 + 3 * 4) # multiplica 3 e 4, e depois adiciona 2
```

```
14
```

In [25]:
```
print((2 + 3) * 4) # soma 2 e 3, e depois multiplica por 4
```

```
20
```

Exercício 1

Sem rodar código algum e considerando as variáveis...

```
quatro = 7
zero = 2
```

...diga: qual o `output` de cada comando abaixo?

```
print(quatro + zero)
print(quatro / zero)
print(quatro // zero)
print(quatro % zero)
print(quatro * zero - quatro / zero)
print(quatro * (zero - quatro) / zero)
```

Exercício 2

Segundo o G1 em 9 de julho, até aquela data 82.908.617 pessoas haviam tomado a primeira dose da vacina contra a covid-19. Especificamente naquele dia, 994.468 pessoas tomaram a primeira dose.

Arredondando, o Brasil tem 212 milhões de habitantes, dos quais cerca de 21% tem menos de 18 anos -- ou seja, não são elegíveis para a vacinação.

1. Quantos brasileiros são elegíveis para a vacinação?
2. Se o ritmo de vacinação da primeira dose se mantiver como no dia 9 de julho, em quantos dias (partindo do dia posterior, dia 10) toda a população elegível terá recebido a primeira dose?

Tarefa

- Vale 0,2 pontos
- Enviar respostas + scripts `.py` para eu@rodolfoviana.com.br até dia 21/07

Segundo a FGV Social, a partir dos microdados da Pnad-C Anual e Pnad Covid, do IBGE, a pirâmide populacional de classes econômicas se mostra da seguinte forma:

período	renda	população
2012	menos de ½ sm	65.800.895
	½ a 1 sm	59.990.268
	1 a menos de 2 sm	44.343.219
	2 a menos de 4 sm	18.125.985
	4 sm ou mais	10.054.567
2019	menos de ½ sm	65.229.668
	½ a 1 sm	61.909.343
	1 a menos de 2 sm	50.078.060
	2 a menos de 4 sm	21.519.066
	4 sm ou mais	11.410.989
jul.2020	menos de ½ sm	52.127.922

período	renda	população
	½ a 1 sm	76.318.115
	1 a menos de 2 sm	56.215.080
	2 a menos de 4 sm	18.646.895
	4 sm ou mais	8.447.679
ago.2020	menos de ½ sm	50.176.044
	½ a 1 sm	76.590.769
	1 a menos de 2 sm	56.859.091
	2 a menos de 4 sm	19.185.258
	4 sm ou mais	8.930.353

(Cabe ressaltar que, como a população cresce a cada mês, trabalhar com números brutos pode induzir a erro. Em alguns casos, é importante trabalhar com proporções -- ou seja, a parte (%) em relação ao total da população do referido período.)

Segundo o release do estudo,

> Levantamento de classes econômicas brasileiras realizado a partir de dados factuais coletados durante a pandemia mostra que o número de pobres no Brasil (renda domiciliar per capita até ½ salário mínimo) caiu 15 milhões entre 2019 e agosto de 2020.

Pergunta 1: Essa informação é verdadeira?

Pergunta 2: Qual foi a queda percentual?

Também segundo o release,

> Já os estratos mais abastados com renda acima de dois salários mínimos per capita perderam 4,8 milhões de pessoas em plena pandemia.

Pergunta 3: Essa informação é verdadeira?

Pergunta 4: Qual foi a queda percentual?

Pergunta 5: Qual foi a variação em cada faixa da pirâmide entre jul.2020 e ago.2020?

Pergunta 6: Comparando ago.2020 com 2012, qual classe econômica teve maior aumento? E qual teve maior queda?