Unidade 1: Iniciação à programação em Python | **Lição 2: Os tipos de dados em Python**

Nesta segunda lição da Unidade 1, descobrirá como utilizar os tipos de dados em Python.

Objetivos:

- Conhecer os diferentes tipos de dados da linguagem Python.
- Formatar um dado numérico

Conheça os tipos de dados utilizados.

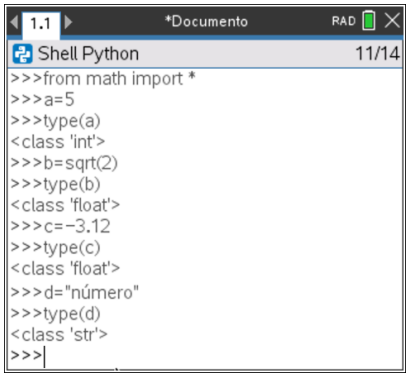
Ao criar um programa em linguagem Python, pode ser necessário conhecer o tipo de variável que está a ser utilizada, ou até modificá-la para uso posterior. Por exemplo, se uma grandeza dada por um programa em Python corresponde a uma grandeza relativa a uma medida em Física, não é necessariamente apropriado manter um resultado com 6 casas decimais.

Vejamos como criar um programa na aplicação **TI-Python** para verificar e distinguir os tipos de grandezas utilizadas:

- uma cadeia de caracteres
- um número real
- um número irracional, $\sqrt{2}$ por exemplo.

Usando a função **type()**, poderá obter o tipo de uma dada variável, por exemplo:

- crie um novo documento Python com uma página Shell Python;
- importe o módulo **maths** (tecla menu, depois opção **4: Matemática** e por fim opção **1: from maths import***);
- defina variáveis, de diferentes tipos, no interpretador;
- utilizando a função **type()** verifique qual a natureza de cada variável que definiu.

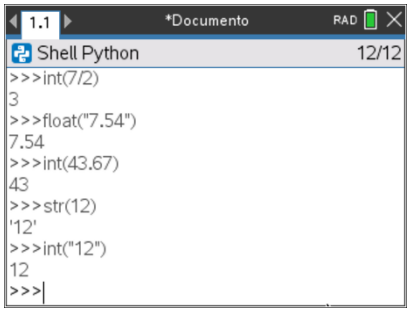


OBSERVAÇÃO:
Todos os comandos usados podem ser introduzidos diretamente com o teclado, escrevendo-os. A função **type()** para determinar a natureza de uma dada variável é digitado manualmente ou inserido através do menu, opção **3:Planos integrados**, depois opção **5: Type** e por fim opção **6: type()**.

DICA:
Ao premir a tecla de direção ▲ seguida de tecla enter pode copiar uma qualquer linha já executada para a linha de edição.

NOTAS:

- O operador **int()** tem como resultado, sempre que possível, o número inteiro representado por uma cadeia de carateres numéricos e a parte inteira de um número compreendido entre $-2\,147\,483\,648$ e $+2\,147\,483\,648$ (codificação de 32 bits, ou seja 4 bytes)
- O operador **str()** transforma um número numa cadeia de carateres.
- O operador **float()** tem como resultado, sempre que possível, o número decimal definido por uma cadeia de carateres.



DICA:
Para incrementar uma variável, dispomos de duas possibilidades:

a) Escrever, por exemplo:

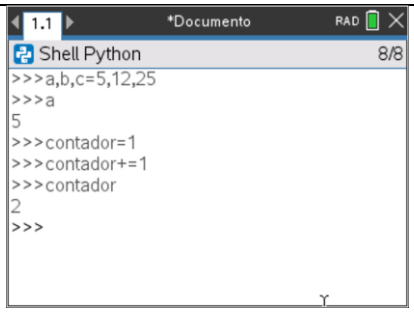
```
contador = 0
contador = contador+1
```

solicitar a exibição da variável

```
contador
1
```

b) Ou então, escrever:
definir o incremento

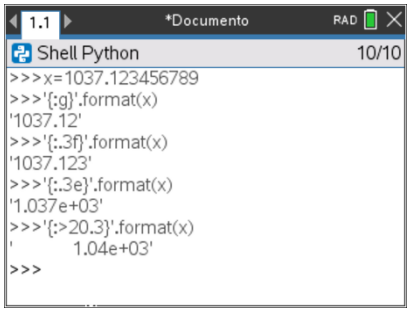
```
contador = 1
contador+=3
contador
4
```



SUGESTÃO:
Utilize, sempre que útil, os habituais atalhos para copiar, **Ctrl** + **C**, colar, **Ctrl** + **V**, e cortar, **Ctrl** + **X**.
Na edição texto, para apagar um carater introduzido erradamente deve clicar na tecla del.

Comentários num programa
Pode inserir comentários nos seus programas em Python, para tal deve colocar o símbolo # no início do comentário, desta forma a respetiva linha não será interpretada no Shell. O símbolo # pode ser obtido clicando na tecla ?!▸.
Poderá ainda usar o atalho ctrl e T para definir uma linha de comando como comentário, o mesmo atalho elimina a referência como sendo um comentário.

MAIS ALÉM:
Formatação de números
O método de formatação em uma *string* é uma ferramenta muito poderosa que permite criar cadeias de carateres substituindo certos campos (colocados entre chavetas) por valores (atribuídos como argumentos da função de formatação) depois de os converter. Pode-se, ainda, especificar dentro de cada chaveta um código de conversão, bem como a forma de apresentação. Vejamos dar alguns exemplos:




```
>>> x=1037.123456789
>>> '{ :g}'.format(x)          # escolha do formato mais apropriado '1.04E+03'
>>> '{ :.3f}'.format(x)        # fixa o número de casas decimais '1037.123'
>>> '{ :.3e}'.format(x)        # notação científica '1.037E+03'
>>> '{0 :20.3f}'.format(x)     # especifica o comprimento da cadeia '       1037.123'
>>> '{0 :>20.3f}'.format(x)    # para justificar à direita '       1037.123'
>>> '{0 :<20.3f}'.format(x)    # para justificar à esquerda '1037.123       '
>>> '{0 :^20.3f}'.format(x)    # centrado '  1037.123   '
>>> '{0 :+.3f} ; {1 :+.3f}'.format(x, -x)  # exibir sempre o sinal   '+1037.123 ; -1037.123'
>>> '{0 : .3f} ; {1 : .3f}'.format(x, -x)  # exibir um espaço se x>0   ' 1037.123 ; -1037.123'
```

A função **.format()**, aplicada a variáveis numéricas tem a seguinte estrutura base:
'{[ordem da variável no argumento da função]:[preencher][alinhar][sinal][largura].[precisão][tipo]}'**.format(**var1,var2,…**)**
cujos respetivos campos explicitam-se abaixo:

- [ordem da variável no argumento da função] - a função .format() pode ter como argumento várias variáveis, sendo que pode ser precedida por várias estruturas base em que este campo defini qual a variável a que se refere a estrutura;
- [preencher] - um qualquer carater que irá preencher os espaços vazios da cadeia de caracteres;
- [alinhar] - definição do alinhamento, sendo: < esquerda; > direita e ^ centro
- [sinal] - define que, caso do valor numérico seja positivo, se apresenta o sinal + , espaço em branco ou nada;
- [largura] - número de carateres da cadeia, isto é, comprimento da cadeia de carateres, podendo alguns serem espaço vazios (comprimento mínimo da cadeia);
- [precisão] - número de casas decimais dependo do tipo de formatação;
- [tipo] - f → decimal, e → decimal em formato exponencial minúsculo, E → decimal em formato exponencial maiúsculo, g → algarismos significativos, b → Binário (apenas argumentos inteiros).