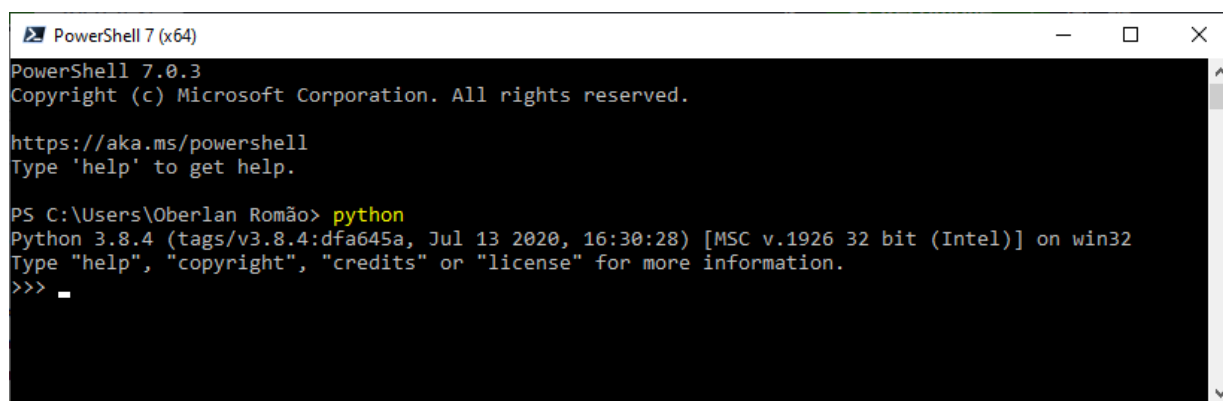




Aula Prática 1

Vamos agora por em prática o que já foi ensinado/aprendido. Entre no Python shell (abra o Terminal ou PowerShell e digite “python”). Você deve ver algo como exibido abaixo:



```
PowerShell 7 (x64)
PowerShell 7.0.3
Copyright (c) Microsoft Corporation. All rights reserved.

https://aka.ms/powershell
Type 'help' to get help.

PS C:\Users\Oberlan Romão> python
Python 3.8.4 (tags/v3.8.4:dfa645a, Jul 13 2020, 16:30:28) [MSC v.1926 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> _
```

Para sair do Python shell você pode digitar “quit()” ou “exit()” ou pressionar as teclas Ctrl + Z (Windows), em seguida, pressionar a tecla Enter. Se, ao digitar “python” no terminal, for exibida uma mensagem dizendo que não reconhece o comando “python”, você pode usar o IDLE ou o Python Shell (instalados ao se instalar o interpretador do Python). Digite IDEL ou Python no menu inicial do Windows.

Crie uma variável com nome “a” e atribua a essa variável o valor 1. Aperte a tecla Enter. Aconteceu alguma coisa? Digite “a” e aperte Enter. O que aconteceu agora? Crie outra variável chamada “b” e aperte Enter. Uma mensagem de erro é exibida. Você consegue entender o que aconteceu? Repita a operação, mas agora atribuindo a variável “b” a expressão “a + 2” (ou seja, b = a + 2. Agora deu certo? Digite “b” e aperte Enter. Qual valor foi exibido?

O nome de uma variável segue algumas regras: só deve conter letras do alfabeto (Python 3.5 aceita acentos e ç), dígitos e underline (_), desde que não comece com um dígito. Além disso, vimos que algumas palavras são reservadas, chamadas de palavras-chaves (keywords). Para visualizar todas as keywords da versão do Python utilizada, você pode fazer o seguinte:

```
1 >>> import keyword
2 >>> print(keyword.kwlist)
```

Obs.: Mais adiante entenderemos o que significa o comando “import keyword”.

Como dito na última aula (remota), toda variável possui um tipo associado. Para que o Python saiba que tipo é esse, ele “olha” o lado direito da atribuição para definir o tipo da variável. Digite “type(a)” para descobrir o tipo da variável “a”. Agora, faça “a = 1.0” e veja o tipo da variável “a”. A variável “a” continuou do mesmo tipo ou mudou? Faça “a = "UFES"” e veja o tipo da variável “a”.

Dica: No terminal, você pode usar a tecla “seta para cima” (↑) para repetir uma operação que já foi digitada.

- Vimos que “a = 1” é legal. E “1 = a”? Por quê?
- E “x = y = 1”? Veja o valor armazenado em x e y.
- Em algumas linguagens, cada instrução termina em um ponto-e-vírgula (“;”). O que acontece se você colocar um ponto e vírgula no fim de uma instrução no Python? Por exemplo, “a = 1;”
- E se você colocar um ponto no fim de uma instrução? Por exemplo, tente fazer “a = 1.”, o que acontece? E se você fizer “a = b.”? Qual a diferença entre esses dois casos?
- Em notação matemática é possível multiplicar x e y desta forma: xy . O que acontece se você tentar fazer o mesmo no Python?

Como você acabou de descobrir, é possível atribuir um mesmo valor a múltiplas variáveis de uma só vez, podemos fazer:

```
1 >>> x = y = z = 100
```

Nesse caso, todas as três variáveis terão o mesmo valor, 100. Além disso, também é possível atribuir múltiplos valores a múltiplas variáveis:

```
1 >>> a, b, c = 5, 3.2, "Hello"
2 >>> print(a)
3
4 >>> print(b)
5
6 >>> print(c)
```

Agora execute o comando abaixo, antes de apertar Enter, o que você acha que vai acontecer?

```
1 >>> nota1, nota2, nota3 = 5.0, 8.2, nota1
```

E se você fizer o comando abaixo?

```
1 >>> nota1 = 5.0; nota2 = 8.2; nota3 = nota1
```

Note que o operador ponto-e-virgula (“;”) é usado para separar as operações que são feitas na mesma linha. O código anterior é equivalente ao código abaixo (que não precisa do “;”):

```
1 >>> nota1 = 5.0
2 >>> nota2 = 8.2
3 >>> nota3 = nota1
```

Salvando o código em arquivos

O Python Shell nos ajuda a testar pequenos códigos e expressões, mas quando precisamos salvar o código para usarmos novamente ou temos um código grande o melhor a se fazer é criar um arquivo com o programa (código). Por estar salvo em um arquivo, esse código pode, facilmente, ser executado e/ou modificado sempre que necessário.

Abra algum editor de texto de sua preferência (como o VSCode ou Sublime Text), crie um arquivo novo e salve-o com o nome `ex1.py` (não se esqueça de adicionar o `.py` no nome do arquivo). Atente-se a localização (pasta) onde você está salvando o arquivo.

Digite no arquivo `ex1.py` o código abaixo:

```
1 nota1 = 5.0
2 nota2 = 8.2
3 nota3 = 6.5
4 print("Nota da primeira prova:", nota1)
5 print("Nota da segunda prova:", nota2)
6 print("Nota da terceira prova:", nota3)
7 print("Nota media do aluno:", (nota1+nota2+nota3)/3)
```

Perceba que no último `print`, foi impresso, junto com uma mensagem, o resultado de uma operação `((nota1+nota2+nota3)/3)`. Isso é interessante quando esse resultado não é usado em outras partes do código. Caso contrário, é mais recomendado criar uma variável para receber esse resultado.

Volte ao arquivo, crie uma variável chamada `media` que recebe o resultado do cálculo `(nota1+nota2+nota3)/3`. Imprima essa variável ao invés da expressão:

```
1 ...
2 media = (nota1+nota2+nota3)/3
3 print("Nota media do aluno:", media)
```

Assim, se precisarmos da nota média do aluno em alguma outra parte do código, podemos usar o valor armazenado na variável “`media`” ao invés de usar a expressão `((nota1+nota2+nota3)/3)`.

Entrada de dados

Até agora os programas criados trabalharam apenas com valores escritos no próprio código. No entanto, o melhor da programação é poder escrever uma solução de um problema e aplicá-la várias vezes. Para isso, é preciso alterar os programas de forma a permitir que novos valores sejam fornecidos durante sua execução, de modo que seja possível executá-los com valores diferentes sem alterar os programas em si.

Chamamos de entrada de dados o momento em que um programa recebe dados (valores/informações) de um dispositivo de entrada (como teclado) ou de um arquivo em disco. A função `input` é utilizada para solicitar dados do usuário. Ela recebe um parâmetro (opcional), que é a mensagem a ser exibida e retorna o valor digitado pelo usuário no formato de string (`str`). *Devemos fazer uma conversão dos dados se quisermos trabalhar com números.* Veja alguns exemplos de utilização do `input`:

1. Simples utilização do `input`

```
1 nome = input("Digite o seu nome: ")
2 print("Ola", nome)
```

2. Ler um valor e convertê-lo para inteiro

```

1 x = input("Digite um numero: ")
2 print(type(x))
3 x = int(x)
4 print(x**2 - 2*x + 3)

```

3. Ler e já converter o valor lido para um `float`

```

1 x = float(input("Digite um numero: "))
2 print(type(x))
3 print(x**2 - 2*x + 3)

```

4. Utilização do `input` sem mensagem

```

1 x = float(input())
2 print(x)

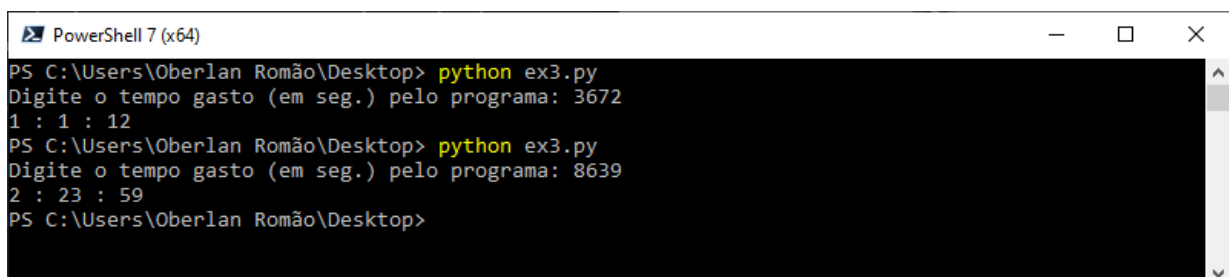
```

Exercícios

1. Modifique o código do arquivo `ex1.py` de modo que as notas das três provas sejam digitadas pelo usuário. Além disso, usando operadores lógicos (não use a cláusula `if`), imprima `True` ou `False` se o aluno precisa fazer prova final (média menor que 7.0).
2. Crie um arquivo chamado `ex2.py` que contenha o código de um programa que calcula o volume e a área de esferas de diferentes raios, dados pelas formulas abaixo. Para isso, crie as variáveis necessárias para que seu código fique o mais genérico possível. O raio deve ser um dado de entrada, ou seja, seu programa deve solicitar o valor do raio ao usuário.

$$V = \frac{4}{3} \times \pi \times r^3 \text{ e } A = 4 \times \pi \times r^2.$$

3. (`ex3.py`) Em computação, é comum precisarmos calcular quanto tempo (em segundos) um programa ficou executando para resolver determinado problema. No entanto, esse medida em segundos, às vezes, não é muito clara e o programador prefere exibir no formato `horas:minutos:segundos`. Por exemplo, se um programa gastou 3672 segundos significa que ele ficou executando 1 hora(s), 1 minuto(s) e 12 segundo(s) (ou 1:1:12). Se o programa gastou 8639 segundos, significa que ele ficou executando 2 hora(s), 23 minuto(s) e 59 segundo(s) (ou 2:23:59). Crie uma variável para armazenar uma quantidade de segundos e, em seguida, converta esse valor para que fique no formato `horas:minutos:segundos`. Veja dois exemplos:



```

PowerShell 7 (x64)
PS C:\Users\Oberlan Romão\Desktop> python ex3.py
Digite o tempo gasto (em seg.) pelo programa: 3672
1 : 1 : 12
PS C:\Users\Oberlan Romão\Desktop> python ex3.py
Digite o tempo gasto (em seg.) pelo programa: 8639
2 : 23 : 59
PS C:\Users\Oberlan Romão\Desktop>

```