Sobre a estrutura básica de um programa em *Python*

Como já citado a diferença entre uma *linguagem de alto-nível* e uma *linguagem de baixo-nível* é que a primeira pode mais facilmente ser "lida" por uma pessoa. Desse modo, é recomendável que você procure sempre deixar o código melhor organizado. Um dos recursos para isso é a utilização adequada de *deslocamento de espaços em branco para iniciar uma linha de comando*, o que é chamado de "indentação". Algo como:if (CONDICAO) : comando1; # note o espacamento nessa linha para indicar que ela esta' subordinada 'a anterior

1. Comentários em meio ao código e o uso adequado de "indentação".

Toda linguagem de programação apresenta o conceito de **comentário**, que serve para indicar ao interpretador *Python* que o texto em *comentário* **não** deve ser *compilado* ou *interpretado*, respectivamente.

Geralmente as *linguagens de programação* oferecem dois modos para comentários, um para *comentar apenas o trecho final de uma linha* e outro para *comentar várias linhas* (ou *bloco de comentários*). Os comentário do primeiro tipo em *Python* usam o símbolo de "sustenido" (`#`), como abaixo:

```
    # ignore o restante desta linha
```

Já os comentários em bloco, precisam de uma marca para abrir uma marca para indicar seu fechamento (nada dentro dele será compilado), como indicado abaixo:

```
""" Ignore todas as linhas entre estas marcas
 print("Esse comando de impressao sera' ignorado!");
 print("Esse tambem...");
 ... """
```

Entretanto, sugiro que **evite** usar esse tipo de comentários, deixando-o reservado para ajudar na depuração do código!
A razão disso é que **não** é possível comentário dentro de comentário, como detalhado na próxima seção.

1.1 Comentários dentro de comentário não é possível em *Python*.

Vale a pena ressaltar a impossibilidade "comentário dentro de comentário", ou seja, se abrir um bloco de comentário """, então a próxima marca """ será interpretada como fechamento da primeira abertura, **não** como abertura para um segundo bloco de comentários.
Nesse sentido é **diferente** dos parênteses em expressões matemáticas, nas quais é perfeitamente válido "parênteses dentro de parênteses", como em "`(2 * (3+4))`".

Para ressaltar a impossibilidade de "comentário dentro de comentário", apresentamos o erro que o interpretador *Python* indicaria. Experimente digitar as 4 linhas seguintes (e depois executar seu código):

"""

```
Esta e' uma tentativa de colocar comentario dentro de comentario, mas NAO
funciona!
""" Ao encontrar o proximo fecha, ele correspondera' ao primeiro abre, """
logo esta linha NAO mais sera' comentario resultando erro de compilacao!
Assim, comentarios encaixados NAO sao permitidos!
"""
```

Cód. 1. Código ilustrando o erro ao supor que existe comentários aninhados em Python.

Ao tentar fazer o computador interpretar o código acima, o interpretador indicará o erro, como indicado na figura 1. Experimente em seu interpretador.

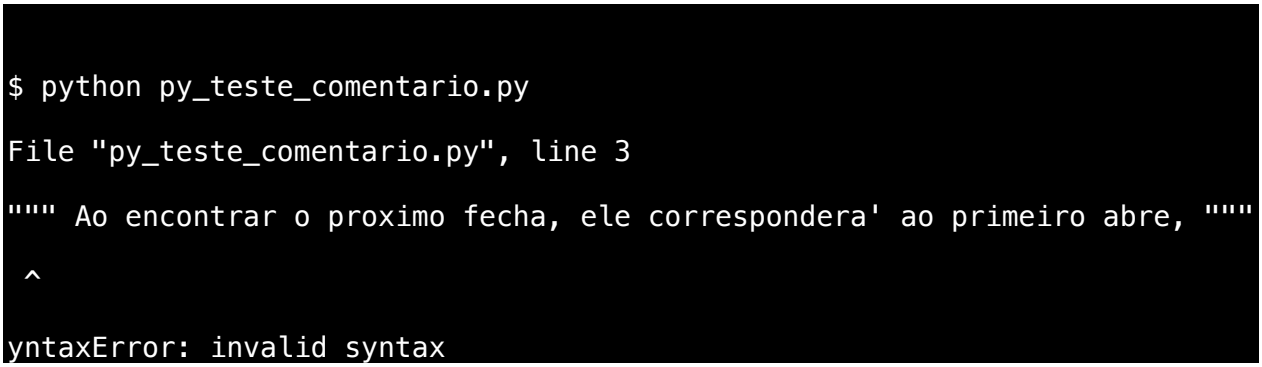

Fig. 1. Ilustração do erro ao supor que existe comentários aninhados em Python.

A razão é que o **aninhamento** de comentários **não** existe na configuração padrão *Python*, assim o segundo """ é interpretado como fechamento da primeira abertura de um bloco de comentários e, desse modo, o trecho entre esse fechamento o próximo """ foi interpretado como código válido, gerando o erro na linha 3 `logo esta linha NAO mais sera' comentario resultando erro de compilacao!`.

Por esta razão recomendo que em seus programas utilize comentários em várias linhas apenas no início do programa (para por exemplo, identificar-se e explicá-lo). Deste modo, fica fácil isolar um bloco grande de seu código (usando comentários em bloco), o que é útil para encontrar erros de sintaxe (mais simples) ou de semântica.

2. Uso adequado de "indentação".

Entenda-se por **"indentação"** o deslocamento horizontal de trechos de código, o que é usado para indicar claramente uma subordinação de comandos (ou **aninhamento**). Por exemplo, se um comando de seleção "`if`" tiver dois comandos subordinados à ele, deve-se "indentar" seus comandos subordinados (isso é obrigatório em *Python*). Por exemplo, podemos usar 2 espaços em branco a mais nesses comandos subordinados. Veja como ficaria esse exemplo:

```
 if (x>0) :
   S += 1; # equivale a: S = S+1;
   print("S=%d" % S);
   # em Python eh obrigatorio a "indentacao" => as 2 linhas acima (e esse
comentario) estao subordinados ao if
```

Cód. 1. Código ilustrativo de comandos subordinados ("indentação").

Para evitar que os códigos fiquem muito "longos" (não cabendo uma linha inteira na tela), sugerimos utilizar apenas 2 espaços para indicar subordinação (como no extrato de código acima).

Assim, se determinado comando começa na coluna *X* ("indentação" de *X* espaços), então todos os comandos subordinados a ele terão deslocamento de ao menos *X+2* espaços em branco. Digo "ao menos", pois dentre estes comandos pode existir outras subordinações. Isso é ilustrado no exemplo abaixo, no qual os comandos das linhas 2 até 5 estão subordinados ao "`repita enquanto`", tendo portanto deslocamento *X+2* ou mais espaços, mas o comando "`print`" da linha 4 tem deslocamento *X+4*, pois ele está subordinado ao "`if`" que já tinha deslocamento *X+2*.

A seguir um exemplo com mais linhas e mais subordinações.

```
      while (cont < N) :
         print("%d" % cont);
         if (cont % 2 == 0) # "cont % 2" e' o resto da divisao inteira,
logo testa se 'cont' e' par"
           print("%d" % cont);
         cont += 1;
```

Cód. 2. Código ilustrativo de comandos subordinados ("indentação").

Note que no código acima, os comandos "`imprima(cont)`" e "`incremente(cont)`" estão deslocados 2 espaços à direita para indicar que ambos estão sobordinados ao comando de repetição "`while (cont < N)`" (laço de repetição será explicado em outra aula). Já o comando da linha 4 está subordinado ao comando da linha 3 ("`if`"), por isso o uso do deslocamento adicional de dois espaços em branco.

4. Estrutura básica de um programa em *Python*.

Abaixo ilustro a estrutura básica de um programa em *Python*, utilizando indentação que deixa o código mais claro e é **obrigatório** em *Python*.
Para tentar, desde cedo, indicar aos aprendizes de programação em *Python* que procurem organizar seus código, sugerimos usar sempre uma função principal, que nominaremos como `main`. Sugerimos que sempre que implementar uma função, faça isso no início de seu código, deixando qualquer declaração de função antes (acima) do trecho que a invoque.

```
"""
 O programa abaixo serve apenas para ilustrar a estrutura basica de um programa
em Python.
 Neste primeiro exemplo existe:
 - Definicao de uma funcao de nome "main": nao e' obrigatorio em Python, mas
usaremos desta forma
```

```
 - Comandos para leitura e para saida de dados: respectivamente, "input" e
"print"
   * Para declarar 2 variaveis que armazenarao valores inteiros, utilizar o "int"
como abaixo
   * Para ler valores (usuario digita) e armazenar na variavel usando a funcao
predefinida "input"
   * O "input" pode ter um argumento que e' uma mensagem (como 'input("Digite:
")'), mas nao usaremos devido
     ao avaliador automatico (voce nao vera as mensagens e elas poderiam provocar
erro na avaliacao)
   * A funcao "int(...)" deve ter como parametro uma expressao numerica,
devolvendo a parte inteira dela
     por exemplo, 'int(3.5+2)' devolvera' o valor inteiro 5
   * O "print" serve para imprimir respostas e pode ter como parametros um
sequencia de "frase" e expressoes,
     por exemplo, supondo que as variaveis num1 e num2 guardem 2 e 3, o comando
         print(num1,"+", num2, "=", num1+num2);
     No Python 3 resultaria na seguinte impressao na tela
         2 + 3 = 5
     Mas no Python 2, resultaria em uma impressao "menos" elegante
         (2, '+', 3, '=', 5)
   * Existe um modo de impressao mais poderoso, que funciona da mesma forma tanto
no Python 2 quanto no 3,:
         print("%d + %d = %d" % (num1,num2,(num1+num2)));
     que sempre resulta na primeira forma acima: 2 + 3 = 5
     Em outra aula sera' explicado o uso de formatadores (como o '%d') para
construcao de tabelas.
"""
def main () : # Pode usar outro nome que nao "main", mas convencionaremos "main"
  num1 = int(input())
  num2 = int(input())
  print(num1+num2)
# note a indentacao dessa linha => NAO pertence ao bloco de comandos da funcao
"main"
main()
```

Leônidas de Oliveira Brandão
http://line.ime.usp.br

Alterações