Comentários, Sintaxe, Variáveis, Tipos de Dados (1), Estrutura de Dados (1), Operadores(1), I/O Básica

Comentários em python

Para saber a funcionalidade de uma variável, linha de código ou outro elemento do código, marcações são essenciais.

Uma das formas mais comuns de se fazer marcações em um código é através de comentários.O ato de comentar seu código, quando feito de forma eficiente, garante muito mais que um bom entendimento. Possíveis manutenções do mesmo serão imensamente facilitadas.

Comentários também são úteis para guardar edições do código, possivelmente necessárias no futuro.

Em python, comentários são feitos utilizando:

- #, para comentar tudo que estiver “à frente” na linha;
- ””””, para comentários de múltiplas linhas;

Observe:

```
1   #Sou um comentario de uma linha so. Isto nao eh impresso
2   print "Agora voce ve"
3   #print Agora nao ve mais
4
5   """
6   Comentario multilinha
7
8   Util para fazer documentacao do codigo
9
10  Ou algum desenho
11
12    M
13  E     S
14    W W
15
16  """
17
18  print "Ive seen things you people wouldnt believe..."
19
```

O papel da identação em python

Você percebeu uma coisa no nosso exemplo do “Hello World!”?

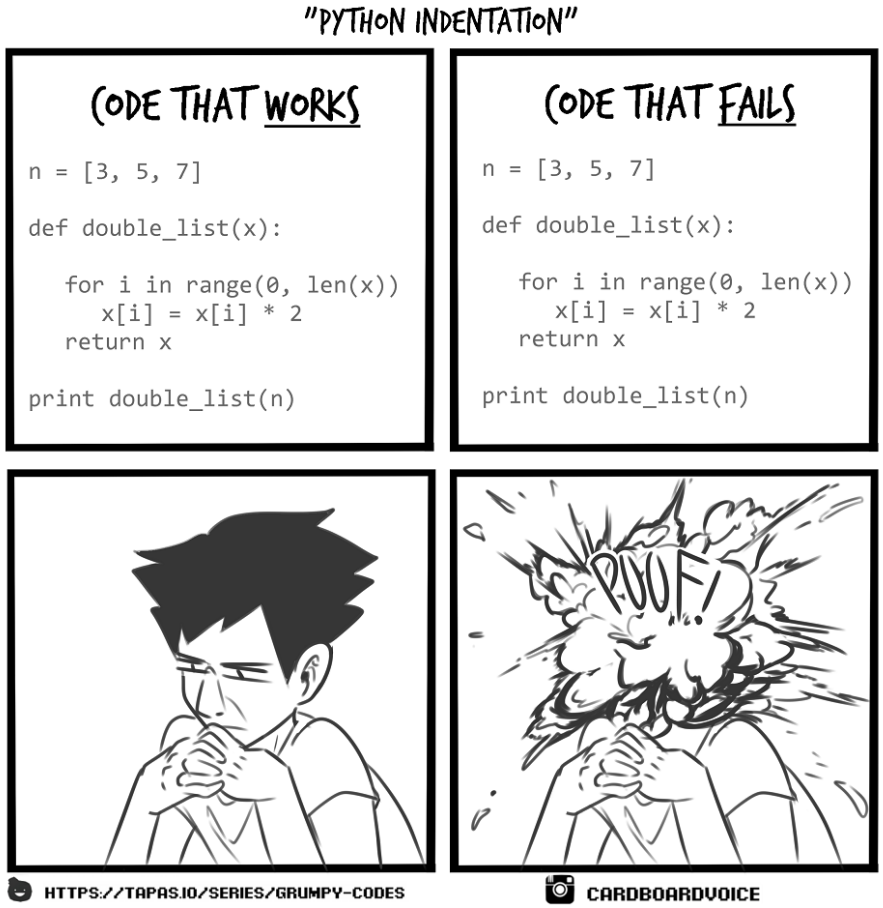
```
1 print "Hello World!"
```

```
1 #include <iostream>
2
3 using namespace std;
4
5 int main(){
6         cout<<"Hello World!"<<endl;
7         return 0;
8 }
```

- Não havia nenhum “caractere especial” para controlar o fim da linha de código, como então o código compilou sem erros?
- A resposta é que, diferentemente de C/C++, Java e outras linguagens, python não utiliza nenhum caractere especial (visível) para controlar a organização seus “blocos de código”. Em vez disso, python utiliza da identação do programa para tal.

- Identação, no contexto de programação, é organizar de alguma forma o código (parte escrita), de forma à seguir um “padrão de organização”. Basicamente, você deixa um “espaçamento correto” aqui, recua umas linhas lá, etc…
- De início, você pode estranhar a forma com que isto acontece, porém a medida que avançar, irá se acostumar com isto. E mesmo que isto não venha a acontecer, ainda existe a opção de usar uma IDE para sua salvação!

- **Veremos ao longo do minicurso, exemplos de como funciona esta questão da identação.**



Sintaxe em python

O computador somente “entende” as instruções que lhe são passadas por conta de que tais instruções já possuem alguma “significância” para o mesmo. Assim, a combinação de tais instruções (estas “palavras”, formando “linhas”) permite ao computador operar aquilo que o programador deseja que seja realizado.

A sintaxe diz respeito ao conjunto de regras que definem como um programa será estruturado, no tocante à suas instruções.

Nosso foco aqui não será definir estes conceitos, visto que os mesmos serão melhor estudados em uma disciplina conhecida como “Linguagens Formais e Autômatos”. O deve ter em mente agora é que em python, existem algumas instruções com um propósito específico (como o já mencionado print).

No entanto, existem em python, algumas instruções que servem à um propósito específico, seja ele declarar alguma estrutura, definir uma função (veremos mais a frente), executar alguma tarefa mais generalista e etc.. A seguir, temos uma lista com estas tais “palavras reservadas” de python.

```
and       del       from      not       while
as        elif      global    or        with
assert    else      if        pass      yield
break     except    import    print
class     exec      in        raise
continue  finally   is        return
def       for       lambda    try
```

Bem como “palavras reservadas”, python também possui alguns símbolos reservados, ou operadores, para as mais diversas funcionalidades. Os mesmos pertencerão à um dos seguintes tipos:

- Aritméticos
- Lógico-Relacionais (Comparação)
- Atribuição
- Lógicos
- Bitwise (Não serão trabalhados neste curso, porém serão explanados no material complementar)
- Associação
- Identidade

```
1  #Exemplos
2  + , - , / , * , ** , % # Aritméticos
3  == , != , < , > , <> , <= , >= # Comparação
4  = , += , -= # Atribuição
5  and , or, not # Lógicos
6  & , ~ #Bitwise
7  in #Associação
8  is #Identidade
```

Variáveis

Quando criamos um programa, muitas das vezes, desejamos “guardar” um valor para que o mesmo possa, repetidamente ser usado. Por exemplo, em um programa de caixa, poderíamos ter um valor guardado para o valor total de compras, bem como muitos outros itens. Para isso, e muitos outras funcionalidades, fazemos o uso das variáveis.

Conceitualmente, variáveis se tratam de locais reservados na memória, onde são guardados algum tipo de valor. Python, bem como a maioria das linguagens de programação, faz uso de variáveis para guarda de informações.

No entanto, o trabalho com variáveis em Python é um tanto quanto distinto, se comparado com outras linguagens, como mostraremos à seguir.

- Declarando uma variável do “tipo inteiro” em Python VS. Em C/C++

```python
inteiro = 42 #Python
```

```cpp
int numero = 42;//C++
```

- Notou uma diferença? Em C/C++, especificamos o tipo antes de declarar o “nome” da variável. O mesmo não ocorreu em Python, porquê?
- Na verdade, variáveis em Python não requerem declaração explícita para a reserva de um espaço de memória. A declaração acontece automaticamente quando um valor é atribuído à variável. Em outras palavras, você não precisa se preocupar com o tipo da variável, ela vai ser o que precisar ser. Prático, não acha?

Assim como em C/C++, o operador “=” indica atribuição (uma variável recebe um valor);

- Uma variável em python é criada através do operador de atribuição.
- Uma variável em python é destruída pelo “garbage collector” (similarmente à como é feito na linguagem Java). Garbage collector é, simplificando, uma entidade que gerencia recursos de memória.

Não há um limite definido para o tamanho do nome de uma variável. Notemos no entanto, que existem algumas restrições para o nome em si de variáveis em python, tais como:

- O nome deve iniciar por letra ou underscore “_”;
- Devemos evitar acentuação;
- Python é Case Sensitive (Caracteres maiúsculos e minúsculos diferenciam o acesso a duas variáveis “homônimas”);

Tipos de Dados

O fato de que os tipos em python não são declarados explicitamente não significa que os mesmos não existam. Todo elemento do código em python possui um tipo.

Isto também podemos elucidar, dadas as capacidades OO de python (para aqueles que já cursaram OO).

Atualmente, python possui os seguintes tipos de dado padrão como sendo seus principais:

- Numéricos:
- String:
- List:
- Tuple:
- Dictionary:

```python
numerico1 = 666 #tipo numerico inteiro

numerico2 = 666.66 #tipo numerico de ponto flutuante

string1 = "Qualquer Palavra" # string

list1 = [1,1,2,3,5,8] # lista numerica

list2 = ["cafe", "guarana", "limonada"] # lista de strings

tupla1 = (2,4,8,16) # tupla numerica

tupla2 = ("abc", "cde") # tupla de strings

Dict1 = {"Javier":"Ganhador", "Ledger":"Ganhador", "Cage":"Ainda nao"} #dicionario
```

Entraremos em mais detalhes sobre cada um conforme avançamos.

Estrutura de Dados

Python em si, simplifica bastante o trato da questão das estruturas de dados, visto que a maioria delas já se encontra de alguma forma, implementada por padrão na linguagem.

Como um exemplo disto, o já citado tipo “List”, que reflete à implementação de estruturas de dados contíguos.

Ainda, como python é OO, a mesma é facilmente “moldável”.

String

Strings em python, tal qual em outras linguagens, são um tipo predefinido para armazenamento de informações literais (texto).

São inicializadas da seguinte forma:

```
1  palavra = "Python"
2
3  john1 = "John Nash"
4  john2 = "John Travolta"
5  john3 = "John Freeman"
6
7  parte_refrao_evidencias = "E nessa loucura de dizer que nao te quero \n Vou negando as aparencias \n Disfarcando as evidencias"
```

```
1  pergunta1 = "Dois parenteses nao eh um pouco exagerado?"# String entre aspas duplas
2  pergunta = 'Quem pegou meus parenteses?' # Strings podem ser desta forma tambem. Nao existe diferenca pratica.
3
4  caractere_simples = "a" #Esta aqui eh um pouco solitaria.
```

Note que, diferentemente de outras linguagens, python não define o tipo “caractere”. Em vez disto, o mesmo é tratado como uma string.

A função “print”, que já tratamos, tem uma relação especial com strings. Tudo que é de saída do console, para fins de impressão, será de alguma forma tratado como string.

Simplificando, para imprimir algo na tela, (seja um número, booleano, texto, etc.), basta usar o “print”.

Uma pequena nota: Você pode ter percebido dos códigos que já foram apresentados, algumas palavras faltavam acentuação (Seja em comentários ou em Strings). A razão para isto é que muitas vezes podem ocorrer “problemas” na codificação dos caracteres, o que resulta em erros na execução dos programas. Este tipo de problema pode ocorrer por diversos fatores. Muita das vezes está relacionado com a forma como python está configurado.

Para resolver isso, uma forma seria usar o “encode” próprio para isso (mais detalhes no slide de “erros”). Não faremos isso, para evitar complicações desnecessárias. Por isso, em nosso minicurso, vamos evitar ao máximo usar caracteres acentuados (Os professores de português ficariam orgulhosos...).

Algumas operações específicas que podemos realizar em strings:

- Concatenação:
- Repetição:
- Presença:

```
1  str_1 = "Bella"
2  str_2 = "Ciao"
3
4  concat = str_1 + " " + str_2 #Concatenacao
5
6  print concat #Retorna "Bella Ciao", concatenado
7
8  print 3 * concat #Repeticao
9
10 print ("Ciao" in concat) # Presenca, testa a presenca do segmento "Ciao" em concat. Retorno Booleano
```

E mais algumas outras que trataremos à frente...

Um detalhe que vale a pena ser citado: A operação de concatenação, apesar de simples, não é muito eficiente quando se trata do uso de memória.

Como recomendação (para alguns casos), temos o uso de “operadores de interpolação”, como no exemplo abaixo. Alguns dos símbolos para este trato estarão disponíveis em materiais complementares.

```
1
2   print "Ve estes %d simbolos diferentes? Estao aqui por algum %s." %(2, "motivo") #Consegue adivinhar a saida?
3   #Ve estes 2 simbolos diferentes? Estao aqui por algum motivo.
```

Caracteres de escape

Existem algumas classes de símbolos especiais no trato de Strings. Os chamados “caracteres de escape” são um exemplo. Tais símbolos não são impressos na tela, porém executam tarefas específicas quando interpretados. Os caracteres de escape mais comuns são os que seguem:

```
1  # \n : newline
2  print "These are the last words \nI'll ever speak \nAnd they'll set me free\n\n"
3  # \t : tab
4  print "Mar \t Vermelho"
5  # \v : tab vertical
6  print "Um tanto \v verticalizado..."
7  # \ : barra simples, trato de caracteres especiais
8  print "Agora posso imprimir \\, meu sonho eh real!"
```

Podemos também, definir strings com “formatação livre”, onde tudo que estiver definido dentro da mesma, será tratado unicamente como string. Ou seja, não serão necessários por exemplo, caracteres de escape. Na prática, isto é útil porém pouco empregado. E sim, a sintaxe é a mesma do comentário multilinha.

```
1   #Texto gerado a partir do mussumipsum.com
2
3   print """
4   Mussum Ipsum, cacilds vidis litro abertis.
5   Detraxit consequat et quo num tendi nada.
6   Todo mundo ve os porris que eu tomo, mas ninguem ve os tombis que eu levo!
7   Interessantiss quisso pudia ce receita de bolis, mais bolis eu num gostis.
8   Nullam volutpat risus nec leo commodo, ut interdum diam laoreet. Sed non consequat odio.
9
10  Per aumento de cachacis, eu reclamis. Quem num gosta di meh, boa gentis num eh.
11  Praesent vel viverra nisi. Mauris aliquet nunc non turpis scelerisque, eget.
12  Ta deprimidis, eu conheco uma cachacis que pode alegrar sua vidis.
13  ///\\\
14  """
15
16  #Texto sera impresso neste devido formato
```

Exercício #2 (5 min.)

Crie as variáveis “nome” e “ocupacao”. Atribua à elas os valores string que desejar. Em seguida, imprima as mesmas na tela, linha a linha.

- Desafio: Imprima as mesmas 10 vezes na tela, porém seu código não deverá ultrapassar 3 linhas.

Numéricos

De maneira similar às strings, variáveis numéricas são criadas quando um número é atribuído às mesmas.

```
1  numero_inteiro = 2501; # Numero Inteiro
2
3  num_float = 9.6; # Numero de ponto flutuante
4
5  num_complex = 5 + 1j # Numero Complexo
```

Python possui suporte à tipos numéricos inteiro (int), inteiros longos (long), ponto flutuante (float) e complexos (complex).

Se tratando da especificidade de cada tipo, caberá ao programador escolher qual usar.

Vale notar que a atenção é necessária no trato de questão de tipo de variáveis numéricas, pois isto pode causar problemas catastróficos, caso ocorra algum erro por parte do programador. Isto vale para toda e qualquer linguagem. De exemplo, o caso Ariane V:

Python ainda possui definidas, em seu módulo matemático, duas constantes numéricas “por padrão”. São elas:

- Pi:
- e (Numero de Euler):

```
1    import math #Sera tratado mais a frente
2    print math.e # e
3    print math.pi # pi
```

Outras formas de atribuição

Já testemunhamos que a atribuição é feita pelo operador “=”, porém, Python (assim como outras linguagens) nos provê com alguns atalhos para o trato de algumas outras formas de atribuição. A exemplo:

```
1  contadora = 0 # Atribuicao comum
2  print contadora # 0
3  contadora += 1 # Atribuicao com incremento, equivalente, nesse caso, a contador = contador + 1
4  print contadora # 1
5  contadora -= 1 # Atribuicao com decremento, equivalente, nesse caso, a contador = contador - 1
6  print contadora # 0
7  # Ha outras formas, que englobam outros operadores. Vamos nos ater a estas por agora
```

Aritmética em Python

Dado que nosso interesse, em grande parte das aplicações que possivelmente desenvolveremos ao longo de nossas vidas, é usufruir da capacidade do computador para operações de cálculo, iremos agora identificar os operadores aritméticos de python.

- Soma:
- Subtração:
- Multiplicação:
- Divisão:

```
1  result_soma = 10 + 90 # Soma #Resutado = 100
2
3  result_sub = 20 - 12 # Subtracao #Resutado = 8
4
5  result_mult = 2 * 2 # Multiplicacao #Resutado = 4
6
7  result_div = 666 / 6 # Divisao #Resutado = 111
8
9  result_div2 = 1.0 / 2.0 # Divisao nao inteira #Resutado = 0.5
10
```

- Módulo:
- Potência:
- Divisão Inteira:

```
result_mod = 666 % 6 # Modulo (resto da divisao) #Resutado = 0

result_mod2 = 5 % 3 #Resutado = 2

result_pow = 5 ** 2 # Potenciacao #Resultado = 25

result_int_div = 1.0 // 2.0 # Divisao inteira #Resultado = 0
```

Exercício #3 (10 min.)

Implemente em python, a seguinte função:

$$Y = X^2 + \frac{10}{Z} + 66$$

- Imprima o valor resultante de Y, para X = 2 e Z = 1
- À seguir, altere a divisão para divisão inteira e imprima o valor resultante de Y, para X = 9 e Z = 3
- Dica: De vez em quando é bom VARIAR...

I/O

Até o momento, todos os códigos que produzimos neste minicurso apenas retornavam resultados ao usuário (output), através da função print.

Neste momento, estaremos trabalhando como receber uma informação do usuário (input).

Python nos provê com duas funções de entrada de dados básica (através do console/terminal). São elas:

- raw_input
- input

- raw_input:

  É a função de entrada de dados simples. “Lê” os dados da linha de entrada e retorna o mesmo (dentro do fluxo do programa) como uma string (sem o caractere “newline”).

  Exemplo:

```
1  resposta = raw_input("Qual seu filme favorito?") # Imprime na tela "Qual seu filme favorito?" e aguarda entrada do usuario
2  # A entrada sera disposta como uma String
3  print resposta
```

Execução:

```
antonio@natsuki:~/Documentos/Ificina/Educacional/Minicursos/MinicursoPythonBásico/Slides/Exemplos/Inputs$ ls
input1.py  input2.py
antonio@natsuki:~/Documentos/Ificina/Educacional/Minicursos/MinicursoPythonBásico/Slides/Exemplos/Inputs$ python input1.py
Qual seu filme favorito?Shin Godzilla
Shin Godzilla
```

Note que, no interior do código, a string de entrada é retornada para o fluxo padrão. Este retorno é armazenado dentro da variável resposta, então, impresso na tela.

- input:

  Assume que o valor de entrada é uma expressão válida de python. O resultado à ser retornado ao fluxo do programa é o resultado da expressão após sua avaliação. Em uma expressão aritmética, por exemplo, será o resultado matemático final.

```
1  resultado = input("Entre com o codigo: ") #Imprime na tela a mensagem "Entre com o codigo" e aguarda o usuario
2  #O codigo da entrada sera uma expressao valida em python, que sera avaliada pelo interpetador. A seguir, sera guardado seu valor em "resultado"
3  #Por exemplo, com a entrada "2+2", teremos que tal expressao sera avaliada e o resultado "4" sera gravado em "resultado"
4  print resultado
```

Exercício #4 (5 min.)

Lembra-se do exercíco #2? Consistia em passar ao programa um nome e uma ocupação.

Refaça o mesmo, agora atribuindo às variáveis “nome” e “ocupação”, entradas de strings feitas pelo usuário.

- Dica: Por simplicidade, é ideal reutilizar o código do próprio exercício #2. Para a entrada de dados, use seus novos conhecimentos.

Dúvidas?
Sugestões?
Críticas?