Mini apostila de Python - Oficina de Vestibulandos

PET Computação - UFPR

September 2016

Sumário

1 Introdução

Esta apostila foi desenvolvida com o o objetivo de auxiliar na codificação de códigos em Python pelos alunos da Oficina de Vestibulandos dos cursos de Ciência da Computação e Informática Biomédica da UFPR.
Python é uma linguagem de programação simples e de fácil entendimento, mas também muito poderosa, se usada corretamente. Neste material sua sintaxe será explicada sem muito aprofundamento, mas de forma a permitir a resolução dos problemas passados em aula.

2 Obtenção e Instalação

Caso esteja sendo utilizado um sistema Linux ou OS X (da Apple), o Python já vem instalado. Para usá-lo basta digitar o comando "python" (sem aspas) no terminal! Já em sistemas Windows, é necessário baixar o interpretador do site https://www.python.org/ No último, é possível usar a ferramenta "IDLE", que é um ambiente de desenvolvimento integrado para Python, que faz o papel de editor de texto e terminal.

3 Variáveis

Os computadores precisam armazenar dados para usar posteriormente. Isto é feito através do uso de variáveis.
Em python, para criar uma variável basta digitar seu nome e atribuir um valor a ela.
Nos exemplos a seguir, criamos a variável "var" e atribuímos à ela o valor 3, e depois criamos a variável "a", com valor 2.

```
var = 3
a = 2
```

Variáveis têm "tipos". O "tipo" de uma variável é o *tipo de dado* que ela representa. Os mais usados são:

- *int* - Um número inteiro;
- *float* - Um número de ponto flutuante (números com vírgula);
- *string* - Uma sequência de caracteres (geralmente são palavras, mas também podem ser formadas por espaços, aspas, vírgulas - enfim, todo tipo de símbolo usado na escrita de textos).

Observe no exemplo abaixo:

```
Inteiro = 29
PontoFlutuante = 12.351
String = "Esta e uma string! :)"
```

A variável que chamamos de "Inteiro" foi inicializada com o valor 29, e é do tipo int, pois armazena um número inteiro. Já a variável "PontoFlutuante" é inicializada com 12,351 (é usado um . (ponto) ou invés de , (vírgula) por que python é uma linguagem essencialmente em inglês, e números fracionários são representados com .(pontos)) Por fim, a variável "String" começa com a frase "Esta é uma string! :)". Repare no modo como a frase é atribuída - entre aspas duplas. Isto é importante, mas não iremos entrar em detalhes por agora.
O valor de cada variável muda conforme são atribuídos valores a ela. Neste exemplo, a variável "variavel" começa com 7, mas seu valor é alterado para 0.

```
variavel = 7
variavel = 0
```

No programa a seguir, com que valor "var" termina?

```
var = 3
var = 40
```

4 Operações Matemáticas

Uma das principais aplicações da computação no dia-a-dia é na resolução de problemas matemáticos. Para isso, existem operadores que podemos usar para fazer vários cálculos. Os mais usuais são os de soma, subtração, multiplicação e divisão. Soma:

```
2 + 3
11.5 + 12.3
```

Subtração:

```
12 - 3
54 - 7.6
```

Multiplicação:

```
2 * 5
7 * 10
```

Divisão:

```
12/4
14/3.0
```

Também podemos usar operações com variáveis, como no exemplo:

```
varA = 2
varB = 3
varA + varB
```

Qual o resultado da operação?

Analise agora o seguinte programa:

```
a = 7
b = 9
resultado = a + b
```

A variável "resultado" é inicializada com o valor resultante da soma de a + b. Portanto, "resultado" começa com o valor inteiro 16.

5 Entrada de Dados

Em vários programas, queremos dar valores diferentes para as variáveis a cada vez que o rodamos. Por exemplo, em uma calculadora, não podemos fazer o seguinte:

```
a = 7
b = 9
resultado = a + b
```

pois toda vez o resultado da operação será o mesmo. Precisamo então de uma forma de "ler" a entrada do programa. A entrada é o dado que o usuário passa para o programa, ou o que ele digita. Em uma calculadora, as entradas seriam os dois valores a ser operados. Para o programa ler a entrada, usualmente é usado o comando "input ()" (palavra para "entrada" em inglês). O programa a seguir lê dois números digitados pelo usuário, "a" e "b", e atribui o resultado de sua soma à variável "resultado".

```
a = input ()
b = input ()
resultado = a + b
```

Este faz as quatro operações que vimos anteriormente:

```
a = input()
b = input ()
soma = a + b
subtracao = a - b
multiplicacao = a * b
divisao = a / b
```

6 Saída de Dados

Agora que já conseguimos ler a entrada de dados, precisamos de uma maneira de devolver alguma coisa para o usuário. Nos programas que vimos antes, as variáveis criadas não eram devolvidas. Nossa "calculadora" faz as operações, mas não mostra os valores resultantes de novo. Para fazer isso, precisamos "imprimir" o valor das variáveis na tela, e usamos o comando "print()" para isso. Olhe nosso código para a calculadora, agora com a saída definida:

```
a = input()
b = input ()
soma = a + b
subtracao = a - b
multiplicacao = a * b
divisao = a / b
print (soma)
print (subtracao)
print (multiplicacao)
print (divisao)
```

Agora, o usuário digita dois valores, e na sua tela aparecem, respectivamente, os valores da soma, subtração, multiplicação e divisão dos valores dados como entrada. Por exemplo, se a entrada for 4 e 2, a variável "a" é inicializada com 4 e a variável "b" com 2. A variável "soma" recebe 6 = (4 + 2), subtração recebe 2 = (4 - 2), e assim por diante. A saída (ou o que é mostrado na tela) é:

6

2

8

2

Ainda precisamos de uma forma de informar o usuário do que cada valor representa, mas não trataremos disto agora.

7 Estruturas de Controle

As estruturas de controle são estruturas que se referem à ordem na qual o código será executado, e geralmente têm a ver com os valores de certas variáveis. Por exemplo, em um programa que calcula a média das notas de um aluno, e tem como saída "Aprovado", SE a média do aluno for maior que 6, e "Reprovado", SENÃO.
Outro caso que que são usadas estruturas de controle é quando se deseja fazer algo ENQUANTO uma condição é verdadeira.

7.1 Desvio Condicional

Os desvios condicionais podem ser chamados também de "Se/ Senão". Nestes desvios, é feita uma COMPARAÇÃO. Caso ela seja verdadeira, damos um certo comando (ou conjunto de comandos). Caso contrário, podemos, ou não, dar outro conjunto de comandos. Por exemplo, no problema a seguir:

```
if (2 < 3):
        print "2 e menor que 3"
else:
        print "2 e maior que 3"
```

A saída (o que será impresso na tela) é, claramente, "2 e menor que 3". Mas, em outros casos, não sabemos quais os valores da comparação. Agora, prestando atenção em como o programa foi escrito:

```
if (2 < 3):
        print "2 e menor que 3"
```

Reparou que há um espaço em branco antes do comando "print"? Este espaço é importante, e o chamamos de tabulação. Ele serve para que o Python entenda que esta parte do código está "dentro" do "if" - ela só acontece se o "if" for verdadeiro. Da mesma forma, em

```
else:
        print "2 e maior que 3"
```

A mensagem só é impressa se o "else" for verdadeiro (ou, em outras palavras, se o "if" for falso). Para aparecer este espaço, usamos a tecla "TAB" do computador (por isso é chamada TABulação)

Veja o programa a seguir:

```
a = input ()
b = input ()
if (a < b):
        print "a e menor que b"
else:
        print "a e maior ou igual a b"
```

Neste caso, a saída depende da entrada. SE a entrada "a" é maior do que a entrada "b", é impresso na tela "a e menor que

b", SENÃO, é impresso que "a e maior ou igual a b". Se as entradas forem
2
3
a condição "$a < b$" é verdadeira, pois "$2 < 3$". Então, a saída será "a e menor que b". Já com
10
5
"$a < b$" é falso, pois "$10 > 5$", então a saída é "a e maior ou igual a b". Repare que, se um número não é menor do que outro, ele pode tanto ser igual quanto ser maior. Mas, e se quisermos saber se a entrada é maior, menor ou igual? Precisamos fazer três comparações. $a > b$, $a < b$ e $a == b$. Não podemos, por exemplo, fazer o seguinte:

```
a = input ()
b = input ()
if (a < b):
        print "a e menor que b"
if (a == b):
        print "a e iguai a b"
else:
        print "a e maior que b"
```

Isto porque cada "SENÃO" é ligado a um, e apenas um "SE" - ou cada "else" é ligado a um "if".
Portanto, o último comando, "else"

```
else:
        print "a e maior que b"
```

é ligado ao "if" que vem logo antes

```
if (a == b):
        print "a e iguai a b"
else:
        print "a e maior que b"
```

e não tem relação com o primeiro "if". Se rodássemos o programa com duas entradas 1 e 2, por exemplo: O primeiro "if" é verdadeiro, pois $1 < 2$, então é impresso na tela "a e menor que b". O código continua e a condição a == b é falsa. Nesse momento, como este "if" é falso, vamos parar no "else", e a saída é "a e maior que b". Mas sabemo que isso não é verdade. Como, então, resolver este problema?
É aqui que entra o "Senão, se". Temos uma condição inicial, "SE". Então temos um "SENÃO", mas queremos comparar outra coisa, então escrevemos "SENÃO, SE". Podemos continuar escrevendo quantos "SENÃO, SE" quisermos, fazendo várias comparações. Em python, o código para "SENÃO, SE" é o "elif", e temos algo assim:

```
a = input ()
b = input ()
if (a < b):
        print "a e menor que b"
elif (a == b):
        print "a e iguai a b"
else:
        print "a e maior que b"
```

Repare que, se um número não é menor nem igual a outro, ele com certeza é maior. Por isso, podemos usar o "else" na última linha, sem fazer mais comparações - temos certeza, neste caso, de que "a" é maior do que "b".

Encadeando "elif"s, podemos fazer algo assim:

```
a = input ()

if (a == 1):
        print "Voce digitou '1'"
elif (a == 2):
        print "Voce digitou '2'"
elif (a == 3):
        print "Voce digitou '3'"
elif (a == 4):
        print "Voce digitou '4'"
else :
        print "Numero diferente de 1,2,3 e 4"
```

7.2 Laço de Repetição

Laços de repetição são também conhecidos por "Enquanto". Nos laços, o que é feito é, parecido com o if, uma COMPARAÇÃO. Mas, ao invés de realizar um comando apenas uma vez, nos laços os comandos são realizados "ENQUANTO" uma determinada condição é verdadeira. ENQUANTO estou com fome, como mais. No momento em que a condição (estar com fome) tornar-se falsa, saímos do laço e paramos de realizar o comando (comer mais). Veja o código a seguir:

```
i = 0
while i<=10:
        print i
        i = i + 1
```

É um código bem simples, que imprime na tela todos os valores entre 0 e 10. Veja como ele também tem um espaço em branco no código dentro do "while" (a tabulação).

A variável "i" é inicializada com 0. O próximo comando é o "while", que, no caso, repete

```
print i
i = i + 1
```

até "i" ser igual a 10.
Podemos dizer que, a cada passo, o que o código faz é:

- Faz a comparação i <= 10, e, se for verdade:
  - Imprime "i"
  - Soma + 1 ao valor atual de "i" , atualizando o valor desta variável
  - Volta para a comparação i <= 10

O que acontece com i é parecido com isso:

valor de i	Código
0	i = 0
1	i = i + 1
2	i = i + 1
3	i = i + 1
⋮	⋮
10	i = i + 1

Imprimindo "i" na tela a cada passo. Veja o próximo código:

```
i = 30
while i > 0:
        print i
        i = i - 5
```

O que ele faz?

8 E Agora?

Os comandos que aprenderemos acabam por aqui, mas você já pode fazer muita coisa! Tentem por exemplo, misturar os laços de repetição (while) com desvios condicionais (if/else).
Faça um programa que, enquanto $a < 10$, imprime a, se $a >= 5$, e imprime $2a$, senão. Ou então, leia uma entrada do usuário e, se for um número > 5, vá diminuindo ele de um em um, enquanto ele for > 5. Brinque com variáveis e operações aritméticas. Aos poucos você descobrirá quanta coisa dá para fazer.

9 Cheat Sheet

Aqui uma tabelinha com os comandos que vimos até agora!

Criar variáveis	NomeDaVariavel = 0
Ler entrada do teclado	input()
Imprimir saída na tela	print()
Desvios condicionais	
Se	if (condicao) :
Senão, se	elif (condicao) :
Senão	else :
Laço de repetição	
Enquanto	while (condicao) :

Não se esqueça de que tanto os desvios condicionais (if, else, elif) quanto o laço de repetição (while) precisam de tabulação (TAB)!