1ST EDITION - 2021

PYTHON PARA INICIANTES

APRENDA A LINGUAGEM DE PROGRAMAÇÃO PYTHON PASSO A PASSO

BY JOHN BACH

Python para iniciantes

Aprenda a linguagem de programação python passo a passo

2021

Por John Bach

Introdução

Python é uma linguagem de programação que é fácil de aprender e um bom exemplo para iniciantes aprenderem programação para iniciantes. Neste livro você encontrará o que precisa para aprender o básico desta língua.

O que é Python?

Uma linguagem de programação inventada por Guido Van Rosem, a primeira versão do Python saiu em 1991.

Python é uma linguagem de programação interpretativa. Se você já viu programação um pouco, você vai saber que esta linguagem tem uma boa estrutura. Os programadores estão sempre procurando a melhor maneira de escrever linhas de código.

Qual é o propósito do Python?

Python é poderoso e simples, autorizando você a escrever softwares muito simples, e tem várias bibliotecas que permitem que você trabalhe em projetos mais complexos.

Web: Hoje em dia, o Python com a estrutura Django é uma das melhores ferramentas para o desenvolvimento de grandes projetos web, principalmente sites web.

Sistemas: Python também é um dos melhores idiomas usados pelos administradores do sistema para criar software para ajudar com tarefas repetitivas e manter o sistema, e se você quiser escrever aplicativos Java usando Python você pode graças ao projeto Jython.

Por que Python?

Python é uma linguagem fácil de entender e seu código também é fácil de ler e, portanto, a melhor escolha para iniciantes que querem aprender programação. É muito conciso e seu código é curto, o que contribui para aumentar a produtividade do programador e reduz a porcentagem de erros no programa e também ajuda a corrigir erros de forma fácil e rápida.

Python também é usado em campos científicos, como a bioinformática. Há bibliotecas disponíveis para fins como biopython.

Há também bibliotecas especiais para a criação de jogos 2D (assim como 3D), e pyGame é um exemplo.

Quem usa Python?

O Google (fundador do Python trabalha para a empresa desde 2005 a 2012), e Yahoo, Microsoft e NASA são todas organizações baseadas em Python e essas empresas são apenas para citar algumas.

A diferença entre Python 2 e Python 3

Houve muitas mudanças quando o Python 3 saiu, e a maior mudança foi mudar a instrução de impressão de:

```
print "Hello"
```

to me:

```
print ("Welcome")
```

Essa mudança causará muitos problemas ao tentar executar arquivos Python 2 no Python 3, especialmente com o uso frequente desta frase em aplicativos. Mas não se preocupe, depois de aprender o básico do idioma, você não terá grandes problemas com as duas versões, e neste tutorial vamos contar com o Python 2.

Instalar python

Instale o Python no Linux ou MacOS

Se o Linux ou Mac OS é o seu sistema operacional, isso é bom porque o Python já está instalado nesses dois sistemas.

Instale python no Windows

Você pode baixar Python no site oficial.

Qual versão eu escolho?

Tente selecionar a versão mais atualizada e estável, e trabalharemos com o Python 3 porque é a versão mais usada.

Capítulo Um

Variáveis em Python

Na lição anterior, aprendemos como instalar python e como trabalhar com ele. Mas antes disso, você tem que aprender a apoiar o árabe, e você tem que entender alguns dos termos usados na programação (que eu usei nesta aula).

Lembrete: Executamos linhas de código diretamente do interpretador Python. Para obter mais informações sobre o intérprete, consulte a lição anterior.

Suporte à língua árabe em Python

Você pode encontrar um erro se tentar executar o comando para imprimir a frase "Hello world" usando letras árabes, então você deve digitar a seguinte linha, antes de escrever qualquer comando contendo palavras árabes:

```
# - * - coding: utf-8 - * -
```

Terminologia

Saída: significa a resposta ou o resultado fornecido pelo intérprete Python. Por exemplo, se você pedir ao intérprete Python para imprimir a palavra Olá, você usaria a seguinte linha:

```
>>> print "Hello"
Hello
```

Aqui chamamos de Olá a saída que o programa retorna (ou seja, o resultado do comando).

Variável: o nome que damos a uma letra ou "palavra" com um determinado valor, por exemplo:

```
>>> name = "me"
```

Aqui criamos a variável nome e atribuímos o valor abdelhadi.

Se quisermos mostrar o valor abdelhadi, basta escrever o nome da variável no interpretador Python:

```
>>> print name
```

Também podemos adicionar uma saudação como esta:

```
>>> print "Hello" + name
```

O código acima é interpretado e suas saídas são:

Olá para mim

Uma variável pode carregar vários tipos de valores, como números, texto, letras...

Definir os valores

Atribuir um valor a um nome específico é um dos fundamentos mais importantes da programação. Este nome é conhecido em programação variável. Aqui está um exemplo:

```
>>> x = 4
>>> x * x
16
```

No exemplo acima, definimos o valor de 4 para a variável x, ou seja, a variável x detém o valor 4. Então podemos usar x em vez de 4 no resto das linhas em nosso programa. Então fizemos o cálculo x * x, que é 4 * 4, e temos 16 como resultado.

Se você tentar usar uma variável não definida anteriormente em seu programa, você terá um erro como este:

```
>>> foo
Traceback (most recent call last):
  File "<stdin>", line 1, in?
NameError: name 'foo' is not defined

>>> foo = 4
>>> foo
4
```

Acima, tentamos usar a variável foo que não definimos, então recebemos uma mensagem do intérprete Python de que a variável foo não existe. Uma vez definido e definido para o valor de "4" podemos chamá-lo sem qualquer problema.

Se você colocar um valor diferente em uma variável que já detém um valor, o valor antigo é substituído pelo novo valor, de modo que a variável mantenha o novo valor.

```
>>> x = 4
>>> x
4
>>> x = 'hello'
>>> x
'hello'
```

No exemplo acima, definimos o valor de "4" para variável x e, em seguida, definimos "olá" para a mesma variável e eventualmente x carrega olá.

Você pode definir mais de um valor para mais de uma variável por vez.

```
>>> a, b = 1, 2

>>> a

1

>>> b

2

>>> a + b

3
```

Trocar os valores de duas variáveis (de modo que a variável detém o valor do outro) é muito fácil em Python:

```
>>> a, b = 1, 2
>>> a, b = b, a
>>> a
2
>>> b
1
```

Ao atribuir um valor em Python, é importante para o lado direito antes da esquerda, de modo que python reconhece os valores no lado direito e, em seguida, atribui-los às variáveis no lado esquerdo do sinal =. O que está à esquerda do sinal = é a variável e à direita dele é o valor.

Comentários

Os comentários são usados na programação para explicar a funcionalidade de uma determinada linha ou para dar uma ideia do seu programa para quem lê seu código. Você não é obrigado a escrevê-los, aqui está um exemplo de comentários:

```
>>> # The next line is to assign a value to a variable
>>> x = 'hello' # This line is a variable value assignment
```

Números

Já aprendemos a lidar com números de forma simples.

```
>>> 42
42
>>> 4 + 2
6
```

Python também suporta decimais.

```
>>> 4.2
4.2
>>> 4.2 + 2.3
6.5
```

Suporta diferentes operações, como adição, diferença e outros cálculos:

+ Add

- The difference

* Beating

/ division

** Exponent (Force)

O restante da divisão

Vamos tentar essas operações em inteiros:

```
# Add
>>> 7 + 2
9

# Operation difference
>>> 7 - 2
5

# Beating
>>> 7 * 2
14
```

```
# Division
>>> 7/2
3


# Exponent
>>> 7 ** 2
49


# The rest of the division
>>> 7% 2
1
```

Se você olhar com cuidado, você verá que o resultado da divisão 7 por 2 é 3 e não 3.5.Isso é porque o código retorna apenas os números corretos ao trabalhar com ele.

```
>>> 7.0 / 2.0
3.5
>>> 7.0 / 2
3.5
>>> 7 / 2.0
3.5
```

Os cálculos podem ser realizados com base em mais de um fator:

```
>>> 7 + 2 + 5 - 3
11
>>> 2 * 3 + 4
10
```

Isso é muito importante para entender como o Python lida com operações aritméticas, onde há prioridades para operações como a seguinte lista, da menor prioridade a uh:

+ -

*** /**

% **

Para entender isso ainda mais, aqui está um exemplo: Quando você executa 2 + 3 * 4 em Python, a primeira operação realizada é 3 * 4, porque * é mais prioridade do que +, e então o valor é adicionado a 2.

```
>>> 2 + 3 * 4
14
```

Podemos contar com parênteses () para priorizar operações, veja o seguinte exemplo:

```
>>> (2 + 3) * 4

20
```

No exemplo acima, colocamos a operação 2 + 3 entre parênteses para dizer ao intérprete Python que esse processo prioritário tem que ser calculado primeiro. Se não colocarmos os parênteses, o intérprete calcula o processo 4 * 3 e, em seguida, adiciona o número 2.

Todas as operações, exceto ** começar da esquerda e depois para a direita.

1 + 2 + 3 * 4 + 5

↓

3 + 3 * 4 + 5

↓

3 + 12 + 5

↓

15 + 5

↓

20

Exercícios

Exercício 1

Quais são as saídas (valores a serem impressos) do seguinte programa:

```
x = 4
y = x + 1
x = 2
print x, y
```

Exercício 2

Quais são os resultados do seguinte programa:

```
x, y = 2, 6
x, y = y, x + 2
print x, y
```

Exercício 3

Quais são as saídas do seguinte programa:

```
a, b = 2, 3

c, b = a, c + 1

print a, b, c
```

Capítulo II

Strings e listas de texto

Na aula anterior, aprendemos a lidar com dados como variáveis e seus tipos, como números e atribuição. Nesta terceira lição, continuaremos a aprender essa língua aprendendo a lidar com listas e cordas de texto.

Lembrete: Códigos prefixados com "<<<" devem ser executados no interpretador Python.

Listas

Listas são uma ótima maneira de lidar com dados em Python. A lista é sobre uma variável que contém mais de um valor. Esses valores podem ser acessados usando o número de cada valor.

Para entender mais, considere que você tem 5 filhos, para que a lista de crianças:

0, Omar

1, Khaled

2, bom

3, Zed

4, Joseph

Em Python, criamos a lista da seguinte maneira:

```
>>> children = ['Omar', 'Khaled', 'Hassan', 'Zaid', 'Youssef']
```

Vamos chamar a lista acima do nome das crianças, e contém cinco elementos, e cada elemento tem seu próprio número, de modo que a contagem começa a partir de zero, por exemplo, se quisermos chamar o filho de "Omar" teremos que chamá-lo de número (ou seja, número 0), e o método de chamar as outras crianças é o seguinte:

```
>>> print 'Come here' + children [0]
Come here Omar
>>> print 'Come here' + children [1]
Come here Khaled
>>> print 'Come here' + children [2]
Come here Hassan
>>> print 'Come here' + children [3]
Come here Zaid
>>> print 'Come here' + children [4]
Come here Youssef
```

Agora, para passar a aplicar os princípios dos menus ao Python, podemos atribuir uma lista a uma variável como esta:

```
>>> x = [1, 2, 3]
```

Você pode criar uma lista de strings de texto em vez de números:

```
>>> x = ["hello", "world"]
```

Você pode combinar diferentes tipos de valores. Este exemplo combina números e sequências de texto:

```
>>> x = [1, 2, "hello," world "]
```

A lista pode conter outra lista:

```
>>> x = [1, 2, "hello," world ", [" another "," list "]]
```

Ou a seguinte maneira:

```
>>> a = [1,|2]
>>> b = [1.5, 2, a]
>>> b
[1.5, 2, [1, 2]]
```

Podemos usar a função len predefinida para medir o comprimento de uma lista (número de componentes da lista):

```
>>> x = [1, 2, 3]
>>> len (x)
3
```

Acessamos itens da lista digitando o nome da variável que detém a lista e, em seguida, o número do item entre os símbolos []:

```
>>> x = [1, 2, 3]
>>> x [1]
2
>>> x [1] = 4
>>> x [1]
4
```

Note que a numeração começa com zero, de modo que o primeiro elemento da lista é o número 0, o segundo elemento é o número 1 e assim por diante.

Você pode criar uma lista com inteiros de um intervalo na função Range. No exemplo a seguir, criamos uma lista com quatro elementos de 0 a 3, uma lista com três elementos entre os números 3 e 6, e então na última linha criamos uma lista de 3 Elementos entre os números 2 e 10 com um aumento de 3:

```
>>> range (4)
[0, 1, 2, 3]
>>> range (3, 6)
[3, 4, 5]
>>> range (2, 10, 3)
[2, 5, 8]
```

A função len também pode ser usada para calcular o número de itens da lista:

```
>>> a = [1, 2, 3, 4]
>>> len (a)
4
```

Você também pode trabalhar com menus com símbolos matemáticos * e + para duplicar ou combinar itens do menu:

```
>>> a = [1, 2, 3]
>>> b = [4, 5]
>>> a + b
[1, 2, 3, 4, 5]
>>> b * 3
[4, 5, 4, 5, 4, 5]
```

Para acessar determinados itens do menu usamos o número do item, observando que a numeração começa de zero a (número de itens -1).

```
>>> x = [1, 2]
>>> x [0]
1
>>> x [1]
2
```

Se você usar um índice errado (numeração), o intérprete Python retorna um erro:

```
>>> x = [1, 2, 3, 4]

>>> x [6]

Traceback (most recent call last):

  File "<stdin>", line 1, in?
```

IndexError: índice de lista fora do intervalo

Você também pode usar numeração negativa para acessar itens do menu do outro para o primeiro (o último item com um valor de -1 e o primeiro item com o número negativo de itens):

```
>>> x = [1, 2, 3, 4]

>>> x [-1]

4

>>> x [-2]

3

>>> x [-4]

1
```

Podemos dissecar uma lista (dividi-la em partes), da seguinte forma:

```
>>> x = [1, 2, 3, 4]

>>> x [0: 2]

[1, 2]

>>> x [1: 4]

[2, 3, 4]
```

Números negativos também podem ser usados para a divisão:

```
>>> x [0: -1]

[1, 2, 3]
```

Se você deixar o primeiro dígito em branco, o valor padrão será zero, e o valor padrão da segunda linha é o número de itens do menu:

```
>>> x = [1, 2, 3, 4]

>>> a [: 2]

[1, 2]

>>> a [2:]

[3, 4]

>>> a [:]

[1, 2, 3, 4]
```

Um terceiro número pode ser usado para determinar a etapa (ou seja, o valor que adicionamos no item atual em comparação com o anterior), que é um por padrão:

```
>>> x = range (10)
>>> x
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> x [0: 6: 2]
[0, 2, 4]
```

Podemos inverter os itens do menu especificando -1 como o valor de incremento da seguinte forma:

```
>>> x = range (10)
>>> x
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> x [:: - 1]
[9, 8, 7, 6, 5, 4, 3, 2, 1, 0]
```

Você também pode alterar os valores do item do menu definindo outro valor:

```
>>> x = [1, 2, 3, 4]
>>> x [1] = 5
>>> x
[1, 5, 3, 4]
```

O operador em dentro pode ser usado para verificar se um item está na lista. Se ele retorna True, significa que o valor está presente.

```
>>> x = [1, 2, 3, 4]

>>> 2 in x

True

>>> 10 in x

False
```

Você pode adicionar outros valores à lista de funções de anexar. No exemplo a seguir, adicionamos (valor 3) para listar a:

```
>>> a = [1, 2]

>>> a.append (3)

>>> a

[1, 2, 3]
```

Cordas de texto

Strings é a técnica usada para escrever textos em Python, que são cadeias de letras (que por sua vez formam frases).

```
Item 0 => M.

Item # 1 => t

Element # 2 => h

Element # 3 => b

Item # 4 => a
```

Observe que os espaços também são calculados em sequências de texto. Por exemplo, a string "Hello" contém 6 elementos (note a distância após mil).

As sequências de texto estão em cotações duplas ou cotações únicas.

```
>>> x = "hello"

>>> y = 'world'

>>> print x, y

hello world
```

Observe que há uma diferença entre aspas duplas e únicas, e pode ser usada de forma intercambiável.

As strings de texto com mais de uma linha podem ser atribuídas a uma variável usando três símbolos, " " ou ", veja o exemplo a seguir (para funcionar bem, é melhor colocar em um arquivo chamado example1.py e executá-lo com example1.py python) : :

```
x = """ "This is a multi-line string

written in

three lines. """ "

print x


y = " 'multi-line strings can be written

using three single quote characters as well.

The string can contain 'single quotes' or "double quotes"' "

print y
```

No exemplo acima, definimos uma sequência de três linhas para variável x de modo que a saída seja quando a variável x é impressa:

Esta é uma sequência multi-linha escrita em três linhas.

Você também pode criar uma sequência de texto de várias linhas adicionando \ n ao final de cada linha, veja o exemplo:

```
>>> x = 'This is a multi-line string \ nwritten in \ nthree lines.'

>>> print x
```

Saídas do exemplo acima:

Esta é uma sequência multi-linha escrita em três linhas.

A função predefinida Python pode ser usada para medir o número de caracteres de uma string. Esta função é chamada de len e pode ser usada da seguinte forma:

```
>>> len ("Abdelhadi")
9
```

As sequências de texto em Python se comportam como menus, de modo que a sequência age como uma lista de vários caracteres, e pode ser indexada (acesso a elementos de string) e segmentada seguindo o mesmo princípio dos menus, veja o exemplo:

```
>>> a = "helloworld"
>>> a [1]
'e'
>>> a [-2]
'l'
>>> a [1: 5]
"ello"
>>> a [: 5]
"hello"
>>> a [5:]
"world"
>>> a [-2:]
'ld'
>>> a [: - 2]
'hellowor'
>>> a [:: - 1]
'dlrowolleh'
```

O operador em dentro pode ser usado para verificar se uma sequência faz parte de outra sequência. No exemplo a seguir, verificamos esse inferno, cheio, e el estão em olá:

```
>>> 'hell' in 'hello'
True
>>> 'full' in 'hello'
False
>>> 'el' in 'hello'
True
```

Quando a saída é Verdadeira, a pequena sequência faz parte da grande sequência de texto.

Existem muitos processos que podem ser aplicados a strings de texto, alguns dos quais são descritos abaixo:

split: separa uma sequência de texto em segmentos separados por qualquer símbolo (desde que esteja na sequência) passamos para esta função, se você não especificar nenhum separador, use split para separar a sequência de texto com base em um espaço branco (ou seja, dividir a frase em palavras), para entender mais quero dizer olhando para o exemplo a seguir.

```
>>> "hello world" .split ()
['hello', 'world']
>>> "a, b, c" .split (',')
['a', 'b', 'c']
```

join: Esta função reflete a função dividida, combinando itens do menu e retornando uma sequência de texto:

```
>>> "" .join (['hello', 'world'])
'hello world'
>>> ','. join (['a', 'b', 'c'])
'a, b, c'
```

tira: Retorna uma sequência de texto com espaços de arrasto omitidos.

```
>>> 'hello world \ n'.strip ()
'hello world'
```

No exemplo acima, o símbolo \ n denota "Nova Linha" para que a seguinte linha imprima as duas primeiras linhas Olá e o segundo mundo:

```
>>> print 'hello \ nworld'
hello world
```

Você também pode passar um valor de texto para despir para que a função retorne uma sequência de texto sem o valor passado.

```
>>> 'abcdefgh'.strip (' abdh ')
'cdefg'
```

substituir: substitui parte da string ou toda a sequência por outro valor:

```
>>> 'Hsoub Academy'.replace (' Academy ',' I / O ')
'Hsoub I / O'
```

Exercícios

Exercício 1

Qual é a saída do programa a seguir?

```
x = [0, 1, [2]]
x [2] [0] = 3
print x
x [2] .append (4)
print x
x [2] = 2
print x
```

Exercício 2

Quantos itens de menu x no exemplo a seguir (não faça manualmente, use o que aprendeu):

```
x = [1, 2, "hello," world "," Hi ", 4, 8, 3, 0," Abdelhadi "," Hsoub Academy "]
```

Exercício 3

Remova o valor "ruim" da seguinte sequência:

```
>>> 'python is awesome bad'
```

Capítulo III

Trabalhando com classes e grupos em Python

Linhas de tupla

Linhas são uma espécie de dados sequenciais como listas, mas linhas são imutáveis. Uma linha consiste em um conjunto de valores separados por uma círgula.

```
>>> a = (1, 2, 3)
>>> a [0]
1
```

Você também pode criar uma linha sem parênteses:

```
>>> a = 1, 2, 3
>>> a [0]
1
```

A propriedade de medir o número de elementos len e a anatomia da classe também é possível, e essas propriedades são aplicadas como fizemos com as listas da aula anterior.

```
>>> len (a)
3
>>> a [1:]
2, 3
```

Uma vez que os parênteses também são usados para adição, você deve criar uma linha de um valor com uma círio extra.

```
>>> a = (1)
>> a
1
>>> b = (1,)
>>> b
(1,)
>>> b [0]
1
```

Você pode combinar duas linhas em uma linha para que a nova linha contenha os elementos da primeira e segunda linha. Por exemplo, no programa a seguir, combinamos linha a e linha b e atribuímos valores à linha c:

```
>>> a = (1,2,3)
>>> a
(1,2,3)
>>> b = ('Hsoub Academy', 'Abdelhadi')
>>> b
('Hsoub Academy', 'Abdelhadi')
>>> c = a + b
>>> c
(1, 2, 3, 'Hsoub Academy', 'Abdelhadi')
```

As linhas também podem conter vários tipos de dados (números, sequências de texto, valores lógicos...) e, como acontece com listas, os itens de texto podem estar em cotações duplas. Veja o exemplo:

```
# This is an example of the ability to assign values of different types
>>> a = ('Hsoub Academy', 'Python', 3, 10, True)
>>> a
('Hsoub Academy', 'Python', 3, 10, True)

# You can also create a list of text strings between double quotes
>>> a = ("Hsoub Academy", "Python", 3, 10, True)
>>> a
('Hsoub Academy', 'Python', 3, 10, True)
```

Se você quiser criar uma linha com um valor duplicado várias vezes, você pode criar uma linha a partir de um item e, em seguida, multiplicar o número de vezes que deseja:

```
>>> ('Academy',) * 5
('Academy', 'Academy', 'Academy', 'Academy', 'Academy')
```

Quando você digita vários valores e os separa com uma círgula sem incluir os elementos com quaisquer sinais de agrupamento, como parênteses e tags [] eles são linhas por padrão, veja o exemplo:

```
>>> print 'Hsoub academy', 5, False, 'Abdelhadi'
Hsoub academy 5 False Abdelhadi
>>> a, b = 1, 2;
>>> print "Value of a and b:", a, b
Value of a and b: 1 2
```

Operações em linhas

Python fornece várias funções de manuseio de classe e mencionamos a função len e como usá-la acima. É hora de falar sobre algumas das outras funções que ajudam nas salas de aula:

Função Cmp para comparar duas linhas, se as linhas forem iguais, o resultado é 0 e, se forem diferentes, o resultado imprime o valor 1 ou o valor -1 dependendo da diferença:

```
>>> a = (1, 2, 3)
>>> b = (1, 2, 3)
>>> cmp (a, b)
0
>>> a = (1, 2, 3)
>>> b = (1, 2)
>>> cmp (a, b)
1
>>> a = (1, 2, 3)
>>> b = (1, 2)
>>> cmp (b, a)
-1
```

A função máxima retorna o maior valor consecutivo:

```
>>> a = ('Hsoub Academy', 'Abdelhadi')
>>> max (a)
'Hsoub Academy'
>>> b = (1, 2, 3)
>>> max (b)
3
```

A função min retorna o menor valor seguido:

```
>>> a = ('Hsoub Academy', 'Abdelhadi')
>>> min (a)
'Abdelhadi'
>>> b = (1,2, 3)
>>> min (b)
1
```

Grupos

As coleções são uma coleção não estruturada de valores únicos que não repetem um valor mais de uma vez.

Você pode criar um grupo vazio e adicionar itens a ele com adicionar. Nota que adicionar só aceita um item de cada vez:

```
>>> x = set ()
>>> x.add (1)
>>> x
set ([1])
>>> x.add (2)
>>> x
set ([1, 2])
>>> x.add ("Hsoub Academy")
>>> x
set (['Hsoub Academy', 1, 2])
```

Para adicionar vários itens ao mesmo tempo, você deve confiar na atualização e observar que os elementos adicionados devem estar dentro []:

```
>>> x = set ()
>>> x.update ([1, 3, 5, "Hsoub Academy"])
>>> x
set (['Hsoub Academy', 1, 3, 5])
```

Para excluir um item de um grupo, você pode usar a removeção para passar o item que deseja excluir:

```
>>> x = set (['Hsoub Academy', 1, 3, 5])
>>> x.remove ("Hsoub Academy")
>>> x
set ([1, 3, 5])
```

Se você quiser excluir todos os itens em um grupo, a sequência clara cuidará dele:

```
>>> x = set (['Hsoub Academy', 1, 3, 5])
>>> x
set (['Hsoub Academy', 1, 3, 5])
>>> x.clear ()
>>> x
set ([])
```

Você pode copiar um grupo e atribuir o copiado a outro grupo copiando. No exemplo a seguir, copiamos o grupo x e designamos o grupo copiado para y:

```
>>> x = set (['Hsoub Academy', 1, 3, 5])
>>> x
set (['Hsoub Academy', 1, 3, 5])
>>> y = x.copy ()
>>> y
set (['Hsoub Academy', 1, 3, 5])
```

Você pode criar um conjunto com vários valores. Observe no exemplo a seguir que o resultado é um conjunto contendo elementos únicos (o elemento 1 foi excluído por ser uma duplicata):

```
>>> x = set ([3, 1, 2, 1])
set ([1, 2, 3])
```

Uma nova maneira de criar grupos no Python 2.7:

```
>>> x = {3, 1, 2, 1}
set ([1, 2, 3])
```

Você pode adicionar um valor a um grupo pela função de ação.

```
>>> x = set ([1, 2, 3])
>>> x.add (4)
>>> x
set ([1, 2, 3, 4])
```

Você pode combinar dois grupos com um coeficiente | Para que o novo grupo contenha ambos os elementos dos dois grupos, no exemplo a seguir criamos o grupo x e depois criamos o grupo y e depois unimos os dois grupos e atribuímos o resultado ao grupo x_y:

```
>>> x = set (['Hsoub Academy', 1, 3, 5])
>>> x
set (['Hsoub Academy', 1, 3, 5])
>>> y = set (["Dyouri", "Abdelhadi"])
>>> y
set (['Dyouri', 'Abdelhadi'])
>>> x_y = x | y
>>> x_y
set ([1, 3, 5, 'Abdelhadi', 'Hsoub Academy', 'Dyouri'])
```

Como nas listas, você pode verificar se há ou não um valor dentro Isso é mais rápido em grupos de listas, mas você só pode ver uma grande diferença se o número de itens for grande.

```
>>> x = set ([1, 2, 3])
>>> 1 in x
True
>>> 5 in x
False
```

Dicionários

Dicionários são semelhantes aos menus. A diferença é que você pode indexar itens dentro de dicionários de qualquer tipo de valor.

Primeiro criamos um dicionário, para que cada valor tenha uma certa chave, por exemplo, a tecla x carrega o valor 1:

```
>>> a = {'x': 1, 'y': 2, 'z': 3}
```

Você pode acessar os valores através das teclas (em vez dos números nos menus):

```
>>> a ['x']
1
>>> a ['z']
3
```

Você também pode criar um dicionário vazio e, em seguida, atribuir chaves e valores:

```
>>> b = {}
>>> b ['x'] = 2
>>> b [2] = 'foo'
>>> b [(1, 2)] = 3
>>> b
{(1, 2): 3, 'x': 2, 2: 'foo'}
```

Um valor-chave pode ser modificado da seguinte forma:

```
>>> a = {'x': 1, 'y': 2, 'z': 3}
>>> a
{'y': 2, 'x': 1, 'z': 3}
>>> a ['y'] = 'Hsoub Academy'
>>> a
{'y': 'Hsoub Academy', 'x': 1, 'z': 3}
```

No exemplo acima, mudamos o valor da tecla y do valor 2 para o valor da Academia Hsoub. Novas chaves e valores também podem ser adicionados:

```
>>> a = {'x': 1, 'y': 2, 'z': 3}
>>> a
{'y': 2, 'x': 1, 'z': 3}
>>> a ['w'] = 'Hsoub Academy'
>>> a
{'y': 2, 'x': 1, 'z': 3, 'w': 'Hsoub Academy'}
```

Você pode excluir itens usando del da seguinte forma:

```
>>> a = {'x': 1, 'y': 2, 'z': 3}
>>> del a ['x']
>>> a
{'y': 2, 'z': 3}
```

Você pode excluir todos os componentes do dicionário seguindo a clara:

```
>>> a = {'x': 1, 'y': 2, 'z': 3}
>>> a
{'y': 2, 'x': 1, 'z': 3}
>>> a.clear ()
>>> a
{}
```

Todo o dicionário também pode ser excluído:

```
>>> a = {'x': 1, 'y': 2, 'z': 3}
>>> a
{'y': 2, 'x': 1, 'z': 3}
>>> del a
>>> a
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'a' is not defined
```

No exemplo acima, tentamos acessar o dicionário A depois de excluí-lo para que o intérprete devolvesse um erro que não existe.

A função retorna todas as chaves em um dicionário. Da mesma forma, você pode devolver todos os valores de um dicionário usando valores. Se você quiser devolver os valores e chaves em um dicionário, você pode usar itens:

```
>>> a.keys ()
['x', 'y', 'z']
>>> a.values ()
[1, 2, 3]
>>> a.items ()
[('x', 1), ('y', 2), ('z', 3)]
```

O loop para para pode ser usado para avançar o dicionário e extrair valores e chaves:

```
>>> for key in a:
... print key
...
x
y
z
>>> for key, value in a.items ():
... print key, value
...
x 1
y 2
z 3
```

A presença de uma chave em um dicionário pode ser verificada usando o operador em ou a função has_key.

```
>>> 'x' in a
True
>>> 'p' in a
False
>>> a.has_key ('x')
True
>>> a.has_key ('p')
False
```

Você também pode confiar tanto no get e setdefault para que o primeiro exiba um valor-chave se ele existir e, se não existir, ele

retorna o valor padrão que é definido como um segundo parâmetro. Veja o exemplo para a compreensão intencional:

```
>>> d = {'x': 1, 'y': 2, 'z': 3}
>>> d.get ('x', 5)
1
```

A linha anterior devolveu o valor (1) embora definimos um valor padrão (5) porque a chave existe e carrega um valor antecipadamente.

```
>>> d.get ('p', 5)
5
```

No exemplo acima, a função get retorna o valor (5) porque a tecla p ainda não existe.

O Setdefaul define um valor padrão para uma chave se ela não existir.

```
>>> d.setdefault ('x', 0)

1

>>> d

{'x': 1, 'y': 2, 'z': 3}

>>> d.setdefault ('p', 0)

0

>>> d

{'y': 2, 'x': 1, 'z': 3, 'p': 0}
```

Dicionários podem ser usados para representar strings de texto entre teclas, onde a chave é substituída pelo seu valor:

```
>>> 'hello% (name) s'% {'name': 'python'}

'hello python'

>>> 'Chapter% (index) d:% (name) s'% {'index': 2, 'name': 'Data
Structures'}

'Chapter 2: Data Structures'
```

Nota sobre as chaves

Uma única chave não pode ser atribuída mais de um valor. Se você tentar atribuir dois valores a mais de uma chave, o último valor é o valor-chave no final:

```
>>> a = {'FirstName': 'Abd', 'LastName': 'Dyouri', 'Job': 'Writer',
'FirstName': 'Abdelhadi'}

>>> a

{'LastName': 'Dyouri', 'Job': 'Writer', 'FirstName': 'Abdelhadi'}

>>> a ['FirstName']

'Abdelhadi'
```

Dois dicionários podem ser combinados em um dicionário atualizando, com o segundo dicionário passando, no exemplo a seguir, criamos um dicionário a e então criamos o dicionário b, então atualizamos o dicionário a, colocando os componentes do dicionário b e finalmente carregando todos os componentes dos dois dicionários:

```
>>> a = {'FirstName': 'Abd', 'LastName': 'Dyouri', 'Job': 'Writer',
'FirstName': 'Abdelhadi'}

>>> a

{'LastName': 'Dyouri', 'Job': 'Writer', 'FirstName': 'Abdelhadi'}

>>> b = {'Website': 'Hsoub Academy', 'Language': 'Arabic'}

>>> b

{'Website': 'Hsoub Academy', 'Language': 'Arabic'}

>>> a.update (b)

>>> a

{'Website': 'Hsoub Academy', 'LastName': 'Dyouri', 'Job': 'Writer',
'Language': 'Arabic', 'FirstName': 'Abdelhadi'}
```

Aplicação sobre dicionários

Vamos criar um aplicativo simples para converter números em palavras em Python, para que cada número tenha uma palavra que você encontrar, abra um arquivo e salve-o como dict.py e escreva o seguinte código, leia os comentários para entender o código:

```
# - * - coding: utf-8 - * -


# Print the application name

print 'Dictionary Version 1.0.0'


# Define the dictionary and assign values to it

dict = {1: 'One', 2: 'Two', 3: 'Three', 4: 'Four', 5: 'Five'}


# Rotate and print dictionary values and keys

for key in dict:

   print key, 'in English is:', dict [key]
```

A saída do aplicativo (após salvar e executar o arquivo) será a seguinte:

```
Dictionary Version 1.0.0

1 in English is: One

2 in English is: Two

3 in English is: Three

4 in English is: Four

5 in English is: Five
```

Você pode modificar o código como quiser, e você pode adicionar mais valores, cabe a você.

Exercícios

Exercício 1

Adicione os valores das duas linhas a seguir e coloque-as em uma linha chamada c:

```
a = ('One', 'Two', 'Three')
b = ('Hsoub Academy', 'Abdelhadi')
```

Exercício 2

Remova o valor HTML do grupo x no seguinte código:

```
x = set (['Ruby', 'Python', 'HTML', 'Perl'])
```

Exercício 3

Aplique suas instruções para criar um programa que exiba cada país árabe e sua abreviação, por exemplo, o Estado de Marrocos abreviado MA e Egito abreviado EGY. Então o resultado do programa é algo que se parece com isso:

Marrocos: MA Egito: EGY

Capítulo IV

Expressões e deslocamentos condicionais (espaços brancos) em Python

Tendo aprendido na aula anterior como lidar com as duas classes, grupos e dicionários em Python, continuaremos a jornada de aprender essa bela língua. Nesta lição aprenderemos como lidar com expressões condicionais, como usar frases condicionais em nossos programas e como fazer uma compensação adequada ao lidar com várias partes do código. Tenha em mente que todos os códigos que começam com a tag <<< devem ser executados no interpretador Python.

Expressões Condicionais

Python nos dá vários parâmetros para comparar valores, e o resultado da comparação é um valor lógico booleano de Verdadeiro ou Falso.

```
>>> 2 <3
False
>>> 2> 3
True
```

Aqui está uma lista de transações disponíveis no Python.

```
== Equals
=! is not equal
Smaller than
<Greater than
=> Less than or equal to
= <Greater than or equal to
```

Esses parâmetros podem ser usados mais de uma vez por linha:

```
>>> x = 5
>>> 2 <x <10
True
>>> 2 <3 <4 <5 <6
True
```

As transações funcionam em cadeias de texto também (a ordem é léxica, ou seja, a ordem das palavras no léxico).

```
>>> "python"> "perl"
True
>>> "python"> "java"
True
```

Valores lógicos podem ser combinados com coeficientes lógicos:

a e b: Verdade apenas se a e b ambos mantêm True.
a ou b: verdadeiro se um dos dois valores é verdadeiro.
não a: verdadeiro apenas se um detém o valor Falso.

Aqui está um exemplo de operadores lógicos:

```
>>> True and True

True

>>> True and False

False

>>> 2 <3 and 5 <4

False

>>> 2 <3 or 5 <4

True
```

Sentença Condicional

A instrução If é usada para executar uma parte do código se a condição for verdadeira:

```
>>> if x% 2 == 0:
... print 'even'
...


even
>>>
```

No exemplo acima, a palavra mesmo é digitada porque o restante da divisão de 42 por 2 é zero (x% 2 == 0), de modo que a condição pode ser atendida. Se a condição não fosse atendida, o intérprete não retornaria nenhum resultado.

Se pode ser usado como condição para executar uma parte separada do código por deslocamento (veja deslocamento de capítulo e espaços abaixo), isso é útil quando seu programa contém muitas frases. Observe que as frases na linha se x% 2 == 0 são deslocadas por quatro espaços brancos.

Mais

Você pode aumentar a instrução If e traduzir a outra cláusula (caso contrário / se não for verdade), que executa o próximo código se a primeira condição for falsa.

```
>>> x = 3
>>> if x% 2 == 0:
... print 'even'
... else:
... print 'odd'
...
odd
>>>
```

No exemplo acima, a primeira condição estava errada porque o resto da divisão por 3 por 2 não é zero, então o programa mudou-se para executar o código após a outra declaração e digitou a palavra estranha.

Elif por atacado

Você também pode adicionar uma instrução elif, que significa "se a condição não for atendida, consulte a seguinte condição":

```
>>> x = 42
>>> if x <10:
... print 'one digit number'
... elif x <100:
... print 'two digit number'
... else:
... print 'big number'
...
two digit number
>>>
```

No exemplo acima, existem duas condições diferentes. Primeiro, se a condição da variável x for menor que 10, o programa imprimirá a afirmação de que o valor da variável x é um número. Se a condição anterior não for cumprida, o programa passará para o próximo, que segue a declaração do Elif. Ou seja, se o valor da variável x for inferior a 100, o programa imprimirá a frase de que o valor da variável x é de dois dígitos e o outro é executado se nenhuma condição for atendida e o valor da variável x for um número grande (porque consiste em 3 dígitos ou mais).

Espaço, distâncias e deslocamento

Os espaços que precedem o código em Python são muito importantes para organizar o código do programa e facilitar a leitura, e é necessário em Python, se você não seguir as leis de deslocamento não funcionará o programa, e este método é usado para separar os códigos pertencentes a uma determinada parte do programa, e o deslocamento é escrevendo o código após um certo número de distâncias , De preferência quatro espaços brancos, veja o exemplo:

A condição é verdadeira?

Sim, a condição é verdadeira.

.... Nenhuma condição não qualificada

Programa concluído

No exemplo acima, as frases "condição sim é verdadeira" e "nenhuma condição não é verdadeira" dependem da frase "A condição é verdadeira?" O termo "programa acabado" é um acompanhamento do programa geral. Você se baseou em pontos para ilustrar a ideia de deslocamento apenas e você tem que substituir os pontos por espaços em arquivos Python.

O código também pode pertencer a um grupo, que por sua vez pertence a outra frase. Para fazer as coisas rirem, pense em uma árvore genealógica composta por pais e filhos:

Avô

Pai 1

..... 1.1 filho

..... Eu tenho 1,2

.... Pai 2

..... filho 2.1

..... Eu tenho 2,2 anos.

O avô tem dois filhos, e cada filho tem dois filhos.

Até agora identificamos um lugar onde o deslocamento é obrigatório, que são as sentenças condicionais. Por exemplo, o seguinte programa:

```
x = 42
if x% 2 == 0:
    print 'even'
print 'Hello World'
```

No exemplo acima, todas as linhas com quatro espaços (ou seja, a frase que imprime a palavra mesmo) são executadas somente quando a condição é atendida, ou seja, uma parte associada à condição no programa.

Se quisermos imprimir uma frase após verificar duas condições, o programa será o seguinte:

A primeira condição é cumprida, então vá para a segunda condição.

A primeira condição não é cumprida, por isso não se mova para a segunda condição.

Aqui está um aplicativo em Python:

```
# the program
x = 42
if x% 2 == 0:
    print 'even'
    if x / 2 == 21:
        print '42 / 2 = 21 Is true '

print 'Hello World'

# director
even
42/2 = 21 Is true
Hello World
```

No exemplo acima, verificamos que o restante do número 42/2 é 0, e como a condição é verdadeira, a palavra ainda foi impressa e o intérprete passou para a próxima condição: 42/2 é 21, e por sua vez, o programa imprime a frase. 42/2 = 21 É verdade Então ele imprimiu a frase Hello World porque não está vinculada a nenhuma condição.

Claro, se a primeira condição não for atendida, o intérprete não passará para a próxima condição e não fará nada mesmo que a condição seja atendida.

Obtenha valores do usuário do programa

Você pode pedir ao usuário para inserir um valor específico e, em seguida, atribuir esse valor a uma variável no programa, por exemplo, para dizer que você quer receber o usuário, primeiro você deve pedir para ele digitar seu nome, em seguida, você imprimir a frase "Olá, Pessoa", de modo que substitui Pessoa pelo valor que o Usuário inseriu, veja o exemplo:

```
>>> person = raw_input ('Enter your name:')
>>> 'Hello', person
```

Se você executar o programa, a saída é a seguinte:

Digite seu nome:

Depois de digitar meu nome e pressionar ENTER, a saída será a seguinte:

Olá Abdelhadi

O que acontece é que o programa pega o valor de entrada e atribui-o à variável pessoa e, em seguida, você pode imprimir a variável para o usuário.

Implementação

Vamos aplicar o que aprendemos e criar um programa simples para agendar entrevistas de emprego:

Solicitamos ao usuário que forneça as seguintes informações:

- Nome do candidato
- O nome do entrevistador
- Hora da entrevista

Em seguida, imprimiremos uma frase contendo todas as informações fornecidas pelo usuário, abriremos um arquivo chamado interview.py e digitaremos o seguinte:

```
applicant = raw_input ('Enter the applicant name:')
interviewer = raw_input ('Enter the interviewer name:')
time = raw_input ('Enter the appointment time:')
print interviewer, "will interview", applicant, "at" time
```

No exemplo acima, primeiro pedimos ao usuário para digitar o nome do requerente e, em seguida, atribuir o valor à variável requerente, em seguida, pedir o nome do entrevistador e atribuir seu valor ao entrevistador variável e, em seguida, obter o tempo de entrevista do usuário e atribuir o valor ao tempo variável, e finalmente imprimir toda a frase Para todas as informações e apresentá-la mais fácil.

Experimente você mesmo, baseie os valores que deseja, olhe para o resultado, jogue o código, adicione outros recursos, como verificar se o candidato tem mais de 18 anos e lembrar que aprender fazendo é melhor do que qualquer outro método.

Aplicativo prático: Crie um programa de login simples

Agora é hora de aplicar o que aprendemos, e para escrever um programa de login simples, o princípio é o seguinte:

O programa recebe o usuário.

Ele é solicitado para informações cadastrais.

O software utilizado indica que o registro foi bem sucedido ou não.

O programa requer informações de login.

O programa verifica se as informações que você inseriu estão corretas.

Se estiverem corretos, o programa imprime uma mensagem de sucesso.

Se estiver errado, o programa imprime uma mensagem de falha.

As variáveis de que dependeremos:

nome de usuário: O nome do usuário

Senha

password_verification: Confirme a senha (apenas para ter certeza de que o usuário digitou a mesma senha e que ele se lembra dela sem problemas).

Em primeiro lugar:

```
print 'Hello User, this is a basic sign up / login Program'
```

Em segundo lugar, peça ao usuário para fornecer os valores das variáveis (nome de usuário: nome de usuário, senha: senha, password_verification: confirmar senha):

```
username = raw_input ('Enter your username please:')

password = raw_input ('Enter the your password please:')

password_verification = raw_input ('Verify password:')
```

Agora vamos verificar se a senha que o usuário inseriu inicialmente é a mesma de quando a senha foi confirmada comparando as duas variáveis senha e password_verification. Se eles tiverem o mesmo valor, significa que o registro é bem sucedido e vamos imprimir ao usuário uma declaração de que o registro foi bem sucedido. Se os valores não forem iguais, imprimiremos uma declaração informando que o usuário e a senha são incompatíveis.

```
if password == password_verification:

   print 'You have been successfully signed up!'

else:

   print 'The password and the password verification don't match! Please
try again '
```

Já completamos o sistema de registro, e devemos passar para a próxima etapa, a fase de login.

Depois de digitar a frase que o usuário registrou com sucesso (veja o código acima) pediremos a ele informações de login inserindo seu nome de usuário e senha, vamos atribuir esses valores recém-inseridos a duas novas variáveis, para compará-los com os valores das duas variáveis antigas, se os valores corresponderem ao que

acabei de digitar I vamos imprimir uma frase informando que o login foi bem sucedido , se não, vamos imprimir uma declaração informando que o usuário inseriu os valores que estão errados e, em seguida, o programa pára. Agora, vamos desenvolver o código acima e fazer as modificações necessárias para acessar:

```
if password == password_verification:
    print 'You have Successfully Signed up! \ n '
    username_sign_in = raw_input ('Enter your username please:')
    password_sign_in = raw_input ('Enter your password please:')
    if username_sign_in == username and password_sign_in ==
password:
        print 'You have Successfully Signed in!'
    else:
        print 'username or password do not match! Please try again! '
else:
    print 'The password and the password verification do not match!
Please try again '
```

Observe o uso do operador lógico e ao verificar se o nome de usuário e senha inseridos (armazenados nas variáveis username_sign_in e password_sign_in) correspondem ao que foi inserido anteriormente. O fator e garante que ambas as condições sejam atendidas. Então terminamos o código pequeno e simples, e este é o código completo:

```
print 'Hello User, this is a basic sign up / login Program'

username = raw_input ('Enter your username please:')

password = raw_input ('Enter the your password please:')

password_verification = raw_input ('Verify password:')

if password == password_verification:

    print 'You have Successfully Signed up! \ n '

    username_sign_in = raw_input ('Enter your username please:')

    password_sign_in = raw_input ('Enter your password please:')

    if username_sign_in == username and password_sign_in ==
password:

        print 'You have Successfully Signed in!'

    else:

        print 'username or password do not match! Please try again! '

else:

    print 'The password and the password verification do not match!
Please try again '
```

Exercícios

Exercício 1

Quais são as saídas do seguinte programa:

```
print 2 <3 and 3> 1
print 2 <3 or 3> 1
print 2 <3 or not 3> 1
print 2 <3 and not 3> 1
```

Exercício 2

Quais são as saídas do seguinte programa:

```
x = 4
y = 5
p = x <y or x <z
print p
```

Exercício 3

Quais são as saídas do seguinte programa:

```
x = 4

y = 5

p = x <y or x <z

print p
```

Exercício 4

O que acontece depois que o próximo código for executado, haverá algum erro? Explique sua resposta.

```
x = 2

if x == 2:

    print x

else:

    print y
```

Exercício 5

O que acontece depois que o próximo código for executado, haverá algum erro? Explique sua resposta.

```
x = 2
if x == 2:
    print x
else:
    x +
```

Exercício 6

O deslocamento no programa a seguir está correto? Se não, corrija o erro.

```
x = 2
if x == 2:
   print x
   if x + 1 == 3:
   print 'x + 1 = 3'
else:
   print x + 2
```

Capítulo V

Loops em Python

Na lição anterior aprendemos como lidar com expressões condicionais, como usar frases condicionais em nossos programas e como fazer um deslocamento adequado ao lidar com várias partes do código em Python, continuaremos aprendendo essa bela língua. Nesta lição, aprenderemos a lidar com loops repetitivos, como um loop para loop e um loop de um tempo. Tenha em mente que todos os códigos que começam com a tag <<< devem ser executados no interpretador Python.

Os loops permitem que você repita o código várias vezes.

Enquanto loop

Enquanto o loop repete um código quando a condição é verdadeira e só pára quando a condição está errada. Para criar um loop de tempo, você deve especificar o número de vezes que é repetido, defini-lo como uma condição e, em seguida, aumentar o valor de uma variável por um, de modo que ele aumenta até que ele atinja o número especificado na condição e pare. Você também pode especificar uma condição de outro tipo para que o loop não esteja relacionado à execução de código um determinado número de vezes, mas a uma determinada condição (um valor variável se torna uma violação de um valor que especificamos)

Digamos que queremos imprimir "Olá" centenas de vezes, é claro que você pode dizer para repetir a seguinte frase cem vezes:

```
print "Hello"
```

Este método está correto, mas custa muito tempo, e o programador sempre acha mais inteligente (faça o máximo de tarefas possível no menor tempo possível). Portanto, não podemos adotar esse método. As linguagens de programação nos oferecem a capacidade de simplesmente repetir com o menor número de linhas.

Por exemplo, podemos imprimir "Olá" dez vezes da seguinte maneira. Note que as frases na palavra enquanto são deslocadas em quatro espaços brancos:

```
>>> i = 0
>>> while i <10:
... print "Hello"
... i = i +1
```

```
Hello
Hello
Hello
Hello
Hello
Hello
Hello
Hello
Hello
Hello
```

Explicação do exemplo acima:

Linha 1: Estabelecemos um valor inicial para a variável i.

Segunda linha: Adicionamos a condição de que o valor da variável seja menor que o número dez, se for igual a 10 paradas do programa.

Terceira linha: nós imprimimos a palavra

Linha 4: Aumente o valor da variável

Na primeira vez que o intérprete lerá o valor da variável, ache igual a 0, e depois verifique se a condição de que o valor é menor que 10 é verdadeira, e que a condição é atendida pela primeira vez (claro que 0 é menor que 10) o programa continuará funcionando, e imprimirá a bem-vinda declaração "Olá" O intérprete então aumenta o valor da variável por um (com o novo valor sendo 1). O intérprete verifica novamente a validade da condição e, como verificador, a frase "Olá" é impressa novamente, e então aumenta o valor da variável por uma vez, de modo que o novo valor da variável i é 2, que por sua vez Menos de 10, a frase é impressa uma terceira vez. Assim, até que o valor da variável atinja o número 10, a condição se torna falsa (porque o número 10 não é inferior a 10, mas igual a ele), e a repetição pára.

Para ilustrar melhor, você pode digitar o valor da variável i em cada iteração, veja exemplo:

```
>>> i = 0
>>> while i <10:
... print "Hello", i
... i = i +1
...
```

Hello 0

Hello 1

Hello 2

Hello 3

Hello 4

Hello 5

Hello 6

Hello 7

Hello 8

Hello 9

Exemplos práticos de usar um loop de while

Exemplo 1: Lista de acesso, itens de grupo ou linha

Existem vários exemplos práticos de uso do loop While, por exemplo, este loop pode ser usado para girar uma lista e imprimir seus elementos, veja o seguinte exemplo:

```
>>> I = 0
>>> children = ['Omar', 'Khaled', 'Hassan', 'Zaid', 'Youssef']
>>> while i <len (children):
... print children [i]
... i = i + 1
...

Omar
Khaled
Hassan
Zaid
Youssef
```

No exemplo acima, criamos uma lista de crianças com cinco elementos e, em seguida, digitamos cada elemento em cada iteração. Definimos o valor da variável i para ser menor do que o número de itens de menu que determinamos com a ajuda da função len, de modo que o programa pára quando o valor da variável atinge o número 5.

Nota: Este é apenas um exemplo de como usar o loop While. O loop for é geralmente usado para tais aplicativos (executando em listas, grupos, linhas...) porque eles são mais flexíveis e, em última análise, cabe a você (você pode usar o método que melhor lhe convém).

Exemplo 2: O programa funciona até que o usuário responda a uma resposta específica

Digamos que você fez uma pergunta matemática ao usuário (por exemplo, "O que é 4 mais 3"), e você quer parabenizá-lo quando você digitar a resposta correta, é simples, e não excede algumas linhas, abre um arquivo e nomeia Answer.py, e coloca as seguintes linhas:

```
number = raw_input ('Enter the Result of 3 + 4:')
if int (number) == 7:
   print 'Congratulation'
```

Usamos a função int aqui para converter o valor inserido pelo usuário em um valor inteiro, o que é útil porque o valor recebido na função raw_input é uma sequência de texto.

Agora que fizemos uma parte do que é necessário para assumir que o usuário entrou na resposta errada, o que pode ocorrer para você agora é imprimir uma frase que diz ao usuário que a resposta está errada e sair do programa, é bom, mas e se quisermos dar ao usuário o direito de tentar novamente , para que imprima O programa diz que a resposta inserida está errada e, em seguida, pede ao usuário para tentar novamente com a possibilidade de inserir um novo valor, podemos fazer isso com a ajuda de outra frase Veja o seguinte exemplo:

```
number = raw_input ('Enter the Result of 3 + 4:')
if int (number) == 7:
   print 'Congratulation!'
else:
   print 'Sorry, Your Answer is not correct, please try again:'
   number = raw_input ('Enter the Result of 3 + 4:')
   if int (number) == 7:
      print 'Congratulation!'
```

Este código é bom, mas é completamente impraticável, ele usa muita redundância que é inútil, e dá ao usuário apenas mais uma tentativa. E se quisermos dar ao usuário mais tentativas?

Você pode dizer que copiamos o código acima e repetimos várias vezes em nosso programa, essa solução é muito ruim porque não atende ao nosso desejo de fornecer um número infinito de tentativas ao usuário (de modo que o programa pare apenas quando o usuário encontrar a resposta correta), e copiando o código e repetindo-o várias vezes A partir do programa longo e difícil de ler.

Para resolver esse problema, podemos simplesmente repetir o código que verifica a resposta toda vez que o usuário responder à violação, o que significa que podemos dizer ao programa para verificar a resposta se ela está correta. A pergunta até que o usuário atinja a resposta correta, o seguinte é um exemplo do código que executa o propósito:

```
number = raw_input ('Enter the Result of 3 + 4:')

while int (number)! = 7:

   number = raw_input ('Wrong answer please try again, enter the Result
of 3 + 4:')

print 'Congratulation'
```

A primeira linha: o programa solicita a resposta do usuário.

Segunda linha: Enquanto o loop verifica se a resposta está errada, se a condição for atendida, o programa passa para a terceira linha, se não for cumprido (ou seja, o valor da variável é 7), o programa passa para a quarta linha.

Terceira linha: O programa diz ao usuário que a resposta está errada e pede que ele tente novamente.

Quarta linha: O programa imprime os parabéns.

Loop Para

O loop For é outro programa que permite a repetição, que é uma técnica usada para girar em torno dos valores da lista também, e é caracterizado pela definição da variável de contagem no início do loop, ou seja, ao contrário do loop While, você não precisa definir a variável antes de começar a escrever o loop. Uma condição é especificada, mas o número de iterações é determinado pelo número de itens da lista atribuída. Veja o exemplo a seguir. Observe que as frases da linha (para contador na faixa (1, 11) são deslocadas por quatro espaços brancos (veja as compensações e a aula de espaços brancos).

```
>>> for counter in range (1, 11):
... print counter, 'Hello World!'
...
```

```
1 Hello World!
2 Hello World!
3 Hello World!
4 Hello World!
5 Hello World!
6 Hello World!
7 Hello World!
8 Hello World!
9 Hello World!
10 Hello World!
```

No exemplo acima, criamos um loop que gira em uma lista que criamos usando a função de intervalo, e atribuímos valores de iteração à variável contador.

Nota: A função de intervalo cria uma lista de números organizados de menor para maior de acordo com os dois primeiros fatores.

```
>>> range (1,11)

[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

Você pode criar uma lista inversa passando um terceiro coeficiente de 1 e invertendo os coeficientes:

```
>>> range (10, 0, -1)

[10, 9, 8, 7, 6, 5, 4, 3, 2, 1]
```

Um loop para para permite girar elementos de um grupo (menus, linhas, strings de texto...) passando o nome variável após a palavra no código. Observe que o para lettre em v strings são deslocados por quatro espaços brancos (ver lição anterior).

```
>>> v = "Hello"

>>> for lettre in v:

... print lettre

...


H

e

l

l

o
```

No exemplo acima, definimos o valor "Olá" para a variável v e, em seguida, usamos o loop para digitar cada letra desta palavra, que definimos para o lettre variável.

Como visto anteriormente, pois também pode ser usado para repetir um código várias vezes, como fizemos com um loop de tempo com a ajuda da função de alcance:

```
for i in range (0,100):
    print i
```

Também podemos usar o loop para passar através de teclas de dicionário para obter cada tecla individualmente, veja o seguinte exemplo:

```
>>> a = {'x': 1, 'y': 2, 'z': 3}

>>> for key in a:
... print key
...

x
y
z
```

Como dissemos anteriormente na Lição 4, você pode obter todas as chaves, valores e itens do dicionário como uma lista:

```
>>> a = {'x': 1, 'y': 2, 'z': 3}
>>> a.keys ()
['x', 'y', 'z']
>>> a.values ()
[1, 2, 3]
>>> a.items ()
[('x', 1), ('y', 2), ('z', 3)]
```

A partir do acima, concluímos que podemos obter cada um dos valores do dicionário:

```
>>> a = {'x': 1, 'y': 2, 'z': 3}
>>> for value in a.values ():
... print value
...


1
2
3
```

Assim, você pode concluir que podemos passar pelas chaves dos dicionários e seus valores juntos ao mesmo tempo:

```
>>> a = {'x': 1, 'y': 2, 'z': 3}

>>> for key, value in a.items ():
... print key, value
...

x 1
y 2
z 3
```

Exemplos práticos de usar o loop For

Exemplo 1: Imprimir chaves e valores para um dicionário específico

Podemos usar um loop para imprimir cada tecla e seu valor em um dicionário.

Por exemplo, digamos que temos um dicionário que contém os nomes das cores e o código de cada cor no sistema hexario (um sistema usado por desenvolvedores web para atribuir uma cor específica a um elemento, em vez de digitar o nome da cor usada como seu código), e queremos imprimir cada cor seguida de seu código , o dicionário que temos é o seguinte:

```
colors = {
    "red": "# f00",
    "green": "# 0f0",
    "blue": "# 00f",
    "cyan": "# 0ff",
    "magenta": "# f0f",
    "yellow": "# ff0",
    "black": "# 000"
}
```

E queremos mostrar essas chaves com seus valores bem, você pode pensar em uma maneira de realizar o que queremos?

A solução é usar um loop para passar cada tecla e seu valor e, em seguida, imprimi-la, podemos fazer isso com apenas duas linhas de código Python, como segue (colocá-lo em um arquivo com uma extensão py e executar o arquivo).

```
colors = {
   "red": "# f00",
   "green": "# 0f0",
   "blue": "# 00f",
   "cyan": "# 0ff",
   "magenta": "# f0f",
   "yellow": "# ff0",
   "black": "# 000"
}
for color_name, hex_value in colors.items ():
   print color_name + ':' + hex_value
```

As saídas do programa serão as seguintes:

```
blue: # 00f
yellow: # ff0
green: # 0f0
cyan: # 0ff
magenta: # f0f
red: # f00
black: # 000
```

Este é apenas um exemplo simples de usar tanto o loop para e dicionários no chão. Você não tem que colocar um fim à sua criatividade. Você pode usar o que aprendeu de diferentes maneiras e programar software que faz funções ilimitadas, então não associe

seu conhecimento a esses exemplos, mas pense em uma maneira de me beneficiar de eu aprendi a produzir programas mais complexos e multifuncionais, e eu ficaria feliz em compartilhar seus programas conosco.

Pare um loop com uma frase: break

Um loop repetitivo de um tipo de tempo pode ser interrompido em um determinado ponto pela quebra e o ponto de interrupção pode ser especificado usando a instrução if, veja o exemplo:

```
>>> i = 0
>>> while i <30:
... if i> 15:
... print "Stop the loop"
... break
... print i
```

```
... i = i + 1
...
```

```
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
```

Pare o loop

No exemplo acima, o loop deveria completar a execução até que a variável eu atingisse o valor de 30, mas parou quando o valor da

variável chegou a 15 porque estabelecemos uma condição para parar a iteração quando eu fiquei maior que 15 e como a condição se tornou verdadeira o programa parou porque pedimos uma quebra por atacado.

Você também pode parar um loop iterativo de um tipo em uma determinada quebra de ponto, e você pode especificar um ponto de interrupção usando a instrução if:

```
>>> list = [1,5,10,15,20,25]
>>> for i in list:
... if i> 15:
... print "Stop the loop"
... break
... print i
...


1
5
10
15
Stop the loop
```

No exemplo acima, a rotação na lista parou após a variável atingir o valor de 15, que é a condição que estabelecemos na instrução if.

Ignorar a execução de código em um loop com uma sentença: continue

Você pode pausar um loop iterativo em um determinado ponto e, em seguida, completar a iteração na próxima etapa, ou seja, passo a passo, usando a instrução continue. Você pode especificar o ponto de interrupção com a instrução if, o que significa que você pode dizer ao programa para mover-se para o próximo se Condição, veja exemplo:

```
>>> list = range (1, 31)
>>> for i in list:
... if i == 15:
... print "Continue the loop"
... continue
... print i
...


1
2
3
4
5
6
7
8
9
10
11
12
13
```

```
14
Continue the loop
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
```

No exemplo acima, o loop deveria completar a execução até que a variável eu atingisse o valor 30, mas parou assim que o valor da variável chegou a 15 e seguiu quando o valor atingiu o número 16 porque estabelecemos uma condição para parar a iteração quando o valor de i é igual ao número 15 e desde a condição O programa parou naquele ponto e imprimiu a demonstração continuar o loop porque pedimos para continuar.

Exercícios

Caro leitor, você deve saber que o aprendizado é completo apenas com a aplicação dos ganhos, então pensar nas soluções desses exercícios é muito eficaz e útil para você, então não seja preguiçoso, não só leia, mas você tem que aplicar o que aprendeu e criar seus próprios programas, especialmente que você aprendeu até agora Você aprendeu uma grande quantidade de técnicas de programação que permitem criar programas impressionantes que fazem muitas coisas.

Você pode postar suas soluções de treino na caixa de comentários se desejar, e é uma boa ideia tentar se exercitar antes de olhar para as respostas de seus colegas.

Exercício 1

Tipo dois programas que imprimem uma frase:

Olá, mundo!

Mil vezes. O primeiro programa usa um loop While e o segundo é um loop For.

Exercício 2

Escreva um programa que pegue um valor do usuário e depois imprima-o 10 vezes.

Exercício 3

Quais são as saídas do seguinte programa:

```
number = 3
list = range (1, 31)
for i in list:
   if i == number:
      continue
   if 15 <i <21:
      continue
   print i
```

Exercício 4

Digite um programa que imprima os seguintes itens do menu com uma instrução de boas-vindas.

```
list = ['Ahmed', 'Abdelhadi', 'Dyouri', 'Hossam', 'Mohammed', 'Khaled',
'Ibrahim', 'Othman']
```

As saídas do programa devem ser as seguintes:

Hello, Ahmed

Hello, Abdelhadi

Hello, Dyouri

Hello, Hossam

Hello, Mohammed

Hello, Khaled

Hello, Ibrahim

Hello, Othman

Exercício 5

Volte para a lição anterior e modifique o programa de login para aceitar a tentativa do usuário novamente se a senha estiver errada.

Capítulo VI

Conheça funções

Até agora, nesta série de lições aprendemos o básico de lidar com tipos de dados, variáveis, frases condicionais e loops para e enquanto. Agora você tem informações suficientes para programar programas intermediários que podem conter centenas de linhas de código. É hora de aprender sobre funções para tornar a tarefa de reutilizar partes do seu código mais fácil e flexível, bem como algumas das funções já definidas em Python.

Como lembrete, todos os códigos que começam com a tag <<< devem ser executados no interpretador Python.

Funções

Uma função na programação como o nome variável pode ser chamada quando necessário, mas a variável contém apenas um valor. A função carrega um código separado, que pode ser uma sequência condicional, uma sequência de impressão, um loop ou todos os acima. As funções são usadas principalmente para evitar duplicar código várias vezes. Por exemplo, você diz que escreveu um código para combinar dois números e então você queria adicionar mais dois números. Assim, se você tentar duplicar o código, será muito cansativo, especialmente se o código que você está repetindo for muito longo, então as linguagens de programação fornecem funções.

Considere digitar "Olá" como o código que a função possui, veja o seguinte exemplo:

```
>>> def say_hello ():
... print "Hello"
...
>>> say_hello ()
Hello
```

Primeiro criamos a função say_hello na linha:

```
def say_hello ():
```

Os três pontos que aparecem após essa linha significam que o intérprete Python está esperando por outras entradas, então digitamos o código de digitação (lembre-se que este é o código base) após um espaço branco (quatro espaços).

Após a criação da função, o intérprete retornará ao seu estado normal, sem qualquer saída, pois devemos chamar a função para executar o código dentro dele, e a chamada é escrita escrevendo o nome da função com parênteses:

```
>>> say_hello ()
```

O que acabamos de ver é como definir uma função, como colocar código dentro dela e, em seguida, como chamá-lo para executar o código. Podemos fazer com que a função devolva um valor sem executar qualquer código (embora execute o código de devolução), veja o seguinte exemplo:

```
>>> def x ():
... return 4
...
>>> x ()
4
```

O que fizemos foi fazer da função um valor em vez de executar um código, ou seja, usamos a função x como uma variável normal com um valor especificado. Você pode se perguntar, qual é o ponto? É muito útil se o seu programa for grande, para que você possa usar funções como um valor de saída após várias operações. Por exemplo, se você tem um programa que pede ao usuário uma entrada (como seu nome ou senha) e você quiser verificar sua entrada e executar operações nele, usar as variáveis aqui não é uma boa ideia porque o código será difícil de modificar. Alternativamente, você pode executar operações na portlet dentro de uma função e, em seguida, fazer a função carregar o valor que você deseja no final.

Nota sobre funções de nomeação

Embora você possa nomear as funções em letras minúsculas e maiúsculas juntas, é uma boa ideia nomear a função em letras minúsculas e separar cada palavra com o caractere "_".

Transações de parâmetros

Uma função da função é que você pode dar-lhe parâmetros quando definido, e dar valores a esses parâmetros, seja quando você definir a função (operadores padrão) ou posterior.

Para entender mais, veja o seguinte exemplo:

```
>>> def say_hello (name):
... print 'Hello', name
...
>>> say_hello ('Abdelhadi')
Hello Abdelhadi
>>> say_hello ('Hsoub Academy')
Hello Hsoub Academy
```

Como você pode ver, definimos a função say_hello com o parâmetro de nome e colocamos uma instrução de impressão dentro dele para imprimir Hello seguido pelo valor do operador de nome. Chamamos a função e atribuímos ao operador de nome dois valores diferentes. Os resultados foram diferentes. Você pode chamar a mesma função com parâmetros diferentes para um número ilimitado de vezes.

Você também pode definir uma função com mais de um parâmetro.

```
>>> def add_numbers (n1, n2):
... return n1 + n2
...
>>> add_numbers (3,4)
7
```

Se definirmos uma função com um parâmetro, devemos definir o valor do operador na chamada ou o intérprete Python retornar um erro da seguinte forma:

```
>>> say_hello ()
TypeError: say_hello () takes exactly 1 argument (0 given)
```

A mesma coisa acontece se você especificar uma série de fatores que devem, por exemplo, dar ao say_hello função um ou mais operadores.

Você pode definir um valor padrão para um parâmetro na linha de definição de função:

```
>>> def num (n1, n2 = 10):
... return n1 + n2
...
>>> num (3)
13
```

Este método pode ser usado para exibir um erro do usuário se esquecer de inserir parâmetros.

```
>>> def say_hello (name = '| Error! Check the line that calls the function'):
... print 'Hello', name
...
>>> say_hello ()
Hello | Error! Check the line that calls the function
```

Você também pode fazer várias operações com funções, aqui está uma lista de algumas dessas operações:

Defina uma função dentro de outra função

Como as funções são partes do código, você pode criar outra função ou uma série de funções dentro de uma função, veja o seguinte exemplo:

```
>>> def say_hello (name):
... def hello ():
... return 'Hello'
... print hello () + name
...
>>> say_hello ('Abdelhadi')
Hello Abdelhadi
```

Como você pode ver acima, o valor Olá é apenas o valor devolvido pela função Hello definida dentro da função say_hello.

Chame uma função dentro de outra função

Este comando é simples, basta chamar a função dentro de outra função e ambas as funções funcionarão quando você chamar a função pai:

```
>>> def say_hello (name):
... print 'Hello', name
...
>>> def print_hello ():
... say_hello ('Abdelhadi')
...
>>> print_hello ()
Hello Abdelhadi
```

No exemplo acima, chamamos a função say_hello de dentro da função print_hello.

Atribua uma função a uma variável

Você pode criar uma função e atribuí-la a uma variável da seguinte maneira:

```
>>> def say_hello (name):
... print 'Hello', name
...
>>> s = say_hello
>>>> s ('Abdelhadi')
Hello Abdelhadi
```

Atribua uma função como parâmetro a outra função

Já dissemos que uma função pode se comportar como variável quando fazemos com que ela devolva um valor de retorno. Isso significa que podemos atribuir outra função à função.

```
>>> def say_hello (name):
... print 'Hello', name
...
>>> def get_name ():
... return 'Abdelhadi'
...
>>> say_hello (get_name ())
Hello Abdelhadi
```

Retorna uma função dentro de outra função

Você também pode usar a instrução de retorno para retornar uma função dentro de outra função em vez de retornar um valor, veja o exemplo:

```
>>> def get_hello ():
... return 'Hello Abdelhadi'
...
>>> def say_hello ():
... return get_hello ()
...
>>> print say_hello ()
Hello Abdelhadi
```

Você notará que a função say_hello detém o valor da função get_hello, pois devolvemos esta última dentro da função say_hello.

Atribua um número infinito de transações a uma função

Digamos que você queira criar uma função que imprima todo o conteúdo (s) que você digita sem ter que especificar cada parâmetro individualmente. Você pode fazer isso com o chamado argumento de comprimento variável:

```
>>> def print_arguments (* args):
... print args
...
>>> print_arguments (2, 4, 8, 'Abdelhadi', 'Hsoub Academy')
(2, 4, 8, 'Abdelhadi', 'Hsoub Academy')
```

A variável args é uma linha tupla de valores, para que você possa lidar com ela com um loop For.

Você pode usar qualquer nome que quisermos para este parâmetro, e geralmente o nome args * é uma tradição entre os programadores Python.

Note que a diferença entre o operando normal e o operando no exemplo acima é a estrela no início.

Você pode fazer dos operadores um dicionário em vez de uma linha para que a função retorne o dicionário que contém o nome e o valor do parâmetro, da seguinte forma:

```
>>> def print_keyword_args (** kwargs):

... print kwargs

...

>>> print_keyword_args (name = 'Abdelhadi', website = 'Hsoub
Academy', n = 3)


{'website': 'Hsoub Academy', 'name': 'Abdelhadi', 'n': 3}
```

Note que o nome do parâmetro começa com duas estrelas desta vez.

Variáveis locais e variáveis globais

Variáveis locais são variáveis que criamos dentro de uma função, e variáveis globais são criadas fora de uma função. Você pode acessar variáveis globais em qualquer lugar do seu programa, ao contrário das variáveis locais que você só poderá acessar dentro da função, veja o seguinte exemplo:

```
>>> def say_hello ():
... name = 'Abdelhadi'
... print 'Hello', name
...
>>> say_hello ()
Hello Abdelhadi
>>> print name
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'name' is not defined
```

Definimos a variável nome dentro da função say_hello e depois tentamos acessá-la fora da função, resultando em um erro que o intérprete Python não reconheceu a variável.

Podemos tornar a variável de nome pública adicionando a palavra global antes dela (você deve fazer isso antes de atribuir um valor à variável), veja o seguinte exemplo:

```
>>> def say_hello ():
... global name
... name = 'Abdelhadi'
...
>>> say_hello ()
>>> print name
Abdelhadi
```

Notamos que conseguimos acessar a variável nome fora da função que foi definida.

Implementação

Vamos dar um exemplo do programa que criamos no estudo de expressões e deslocamentos condicionais.

```
print 'Hello User, this is a basic sign up / login Program'

username = raw_input ('Enter your username please:')

password = raw_input ('Enter the your password please:')

password_verification = raw_input ('Verify password:')

if password == password_verification:

   print 'You have Successfully Signed up! \ n '

   username_sign_in = raw_input ('Enter your username please:')

   password_sign_in = raw_input ('Enter your password please:')

   if username_sign_in == username and password_sign_in ==
password:

      print 'You have Successfully Signed in!'

   else:

      print 'username or password do not match! Please try again! '

else:

   print 'The password and the password verification do not match!
Please try again '
```

Vamos nos concentrar no código de verificação do usuário e na senha:

```
    if username_sign_in == username and password_sign_in ==
password:
        print 'You have Successfully Signed in!'
    else:
        print 'username or password do not match! Please try again! '
```

Se quisermos usá-lo mais de uma vez em nosso programa, temos que colocá-lo dentro de uma função.

```
def user_logged_in? (username_sign_in, username, password_sign_in,
password):
    if username_sign_in == username and password_sign_in ==
password:
        return True
    else:
        return False
```

Você pode usar a função em qualquer lugar do código:

```
def is_user_logged_in (username_sign_in, username, password_sign_in,
password):
   if username_sign_in == username and password_sign_in ==
password:
      return True
   else:
      return False


new_password = raw_input ('Enter new username:')
new_username = raw_input ('Enter new password:')
print 'Ok!'
password = raw_input ('Hello again, Enter your username:')
username = raw_input ('Enter username \' s password: ')


print is_user_logged_in (username, new_username, password,
new_password)
```

A função is_user_logged_in compara as senhas e nomes de usuário se elas forem idênticas, o valor será Verdadeiro.Se os valores não coincidirem, a função é Falsa.Agora você pode usar a função com instruções condicionais em qualquer lugar do seu programa sem ter que reescrever o código de verificação todas as vezes. Além disso, a modificação do código é muito simples, a modificação de um lugar (função) é suficiente para ser aplicada a todos os outros lugares (locais de chamada da função).

Funções predefinidas

Python fornece funções predefinidas, e nós lidamos com muitas delas em aulas anteriores. Aqui estão algumas funções úteis que já estão disponíveis no Python.

impressão: A função de impressão que usamos mais de uma vez para imprimir valores em aulas anteriores.

Int: Uma função que converte valores em um inteiro, por exemplo:

```
>>> int (5.5)
5
```

str: Função para converter valores em uma string de string de texto:

```
>>> str (True)
'True'
```

É frequentemente usado para adicionar uma sequência de texto de outro tipo com o operador + (se você não usar esta função o interpretador Python retornará um erro que você não pode adicionar dois valores de dois tipos diferentes):

```
>>> print 'ABC' + str (122) + str (False)
ABC122False
```

abs: função para obter o Valor Absoluto de um número (se positivo é o mesmo número, e se negativo é o oposto):

```
>>> abs (3)
3
>>> abs (-3)
3
```

len: Uma função para medir o número de itens da lista ou o número de caracteres de uma sequência de texto.

```
>>> a = ['Abdelhadi', 'Dyouri', 'Academy', 'Hsoub']
>>> b = 'Hello'
>>> len (a)
4
>>> len (b)
5
```

min: Uma função para determinar o menor valor entre dois ou mais valores.

```
>>> min (4, 5, 7, 88, 3, 2)
2
```

max: Uma função para determinar o menor valor entre dois ou mais valores.

```
>>> max (4, 5, 7, 88, 3, 2)
88
```

intervalo: Para gerar uma lista de números entre o valor do primeiro parâmetro e o valor do segundo parâmetro:

```
>>> range (0, 11)
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
>>> range (0, 11, 3)
[0, 3, 6, 9]
```

raw_input: função Para obter uma entrada de texto do usuário, você pode atribuir esta entrada a uma variável.

```
>>> def say_hello (name):
... print 'Hello', name
...
>>> say_hello (raw_input ())
Abdelhadi
Hello Abdelhadi
```

Como dissemos anteriormente, você pode atribuir uma função como parâmetro para outra função, que é o que fizemos no exemplo

acima, como passamos a função raw_input para ser um parâmetro para a função say_hello.

Exercícios

Exercício 1

Escreva um programa para obter um valor do usuário e converta-o em um valor numérico.

Exercício 2

Digite uma função para imprimir uma string várias vezes e deixe que o usuário escolha a frequência (ajuda: use raw_input para obter um valor do usuário)

Exercício 3

Digite uma função para combinar dois números inseridos pelo usuário.

Exercício 4

Programe uma calculadora simples que adiciona, subtraia, multiplica e divide. É assim que funciona:

Get the value of the first issue of the user.

Get the value of the second number.

Convert the two text values into numerical values.

Get the mathematical operator (*, /, - or +) from the user.

Use conditional sentences to perform the appropriate operation (if input is + add