

INTRODUÇÃO À LINGUAGEM DE PROGRAMAÇÃO Python

Objetivos

- Apresentar a descrição da linguagem Python;
- Apresentar as estruturas básicas de controle em Python;
- Apresentar a forma de codificação em linguagem Python;
- Apresentar padrões de mapeamento para a linguagem Python.

Histórico

- 1989 – início do desenvolvimento por Guido van Rossum na CWI, Holanda;
- 1991 – publicação da primeira versão aberta do código (*open-source*);
- 1994 – lançamento da versão 1.0;
- 2000 – lançamento da versão 2.0;
- 2008 – lançamento da versão 3.0.

Descrição da linguagem

- Alfabeto

Um programa em Python poderá conter os seguintes caracteres:

- as vinte e seis (26) letras do alfabeto inglês (sem acentos e cedilha):
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
a b c d e f g h i j k l m n o p q r s t u v w x y z

- os dez (10) algarismos:
0 1 2 3 4 5 6 7 8 9

- os símbolos:

| | | | |
|---|------------------------------|-----|------------|
| < | menor | () | parênteses |
| > | maior | [] | colchete |
| . | ponto | { } | chaves |
| , | vírgula | + | soma |
| : | dois pontos | - | subtração |
| ; | ponto-e-vírgula | * | asterisco |
| = | igualdade | / | barra |
| ! | exclamação | # | sustenido |
| ? | interrogação | " " | aspas |
| & | ampersete ("e" comercial) | ' ' | apóstrofo |
| ^ | circunflexo | % | porcento |
| | barra em pé | ~ | til |

- Pontuação
 - Ponto-e-vírgula é usado para separar comandos, na mesma linha; desnecessário, para comandos em linhas diferentes.
 - Em alguns casos de operadores, convém o uso de espaços em branco antes, e depois.
- Observação:
Em Python utilizam-se, **obrigatoriamente**, as letras minúsculas para os comandos próprios da linguagem.

Tipos de dados

- Constantes

- Constante inteira

Exemplos:

10, 532, -10567
 067, 05421, 0724 (octal)
 0L, 1300000000L, 3241L (inteiro longo)
 0x41, 0xFFFF, 0xA0 (hexadecimal)

Observação:

O maior valor inteiro representável é 2147483647.

- Constante real

Exemplos:

10.465 -5.61 +265. 0.0 .731 .37e+2 -3.e-1

Observações:

A vírgula decimal é representada por ponto decimal.

Em geral, tem-se a faixa de 10^{-38} a 10^{+38} .

- Constante literal

Exemplos:

Caractere : '1', " ", '*', 'A', "a", '?'
 Cadeia : "BRANCO", 'CEU AZUL'

Observações:

O tamanho da cadeia é limitado.

As cadeias são terminadas pelo símbolo especial '\0'.

- Caracteres predefinidos:

| | |
|--------|-------------------------------------------------------|
| '\0' | nulo (fim de cadeia de caracteres) |
| '\n' | passa para a próxima linha |
| '\t' | passa para a próxima coluna de tabulação (9,17, ...) |
| '\b' | retorna o cursor uma coluna |
| '\r' | posiciona o cursor no início da linha |
| '\f' | limpa a tela ou passa para a próxima página |
| '\ ' | barra invertida |
| '\'' | apóstrofo |
| '\nnn' | representação de um byte , em octal |
| '\xnn' | representação de um byte , em hexadecimal |

- Tipos básicos de valores constantes

| Algoritmo | Python |
|--------------|--------------------|
| inteiro | 0 |
| real | 0.0 |
| caractere | '0' |
| lógico | True, False |
| (indefinido) | None |

- Variáveis

- Nome de variável

- O nome de uma variável tem tamanho determinado;
- O primeiro caractere é uma letra ou travessão (_);
- Outros caracteres podem ser letra, algarismo ou travessão (_).

Exemplos:

Nomes válidos : l, a, de, V9a, Lista_Notas

Nomes inválidos: x+, t.6, 43x, so 5

- Tipos de valores armazenados

As variáveis terão seus tipos definidos mediante atribuição.

Forma geral:

```
<nome> = <valor>
<nome> = None
```

Exemplos:

```
#para definir:
null = None;
y = 0;           // inteiro
x = y;           // inteiro;
i = 0.0; j = 0.0; k = i; // real
a = 'a'; b = '0'; c = '_'; // caractere
b0 = False; b1 = True; // lógico
s1 = 'abc'; s2 = "abc"; // cadeia de caracteres
```

```
#para exibir:
print 'x=', x
print "y=", y
print "i=", i, ' j=', j, " k=", k ;
print "a="+a+" b="+b+" c="+c+" ";
print ( b0, b1 );
print "s1="+s1+' s2='+s2+'\n';
```

- Variáveis agrupadas

- Listas

Listas são agrupamentos de valores separados por vírgulas (indicados por colchetes).

Forma geral:

```
<lista1> = [valor1, valor2, ... ];
```

Exemplos:

```
frutas = [ "abacate", "abacaxi", "ameixa", "amora" ],  
letras = [ 'a', 'b', 'c' ],  
valores = [ 1, 2, 3, 4, 5 ];
```

#acesso

```
primeira_fruta = frutas[ 0 ];
```

```
ultima_fruta = frutas[ len(frutas)-1 ];
```

Observação:

As listas podem agrupar quaisquer tipos de valores.

O primeiro elemento tem índice igual a zero.

O último elemento tem índice igual ao tamanho da lista menos 1.

- Tuplas e sequências

Tuplas são agrupamentos de valores separados por vírgulas (indicados por parênteses).

Forma geral:

```
<nome> = ()  
<nome> = <valor1>, <valor2>  
<nome> = (<valor1>, <valor2>)  
<nome> = <valor1>, <tupla>  
<nome> = <tupla>, <valor2>  
<nome> = <tupla1>, <tupla2>  
<nome> = (<tupla1>, <tupla2>)
```

Exemplos:

#tupla vazia

```
t0 = ( );
```

#tupla unitaria

```
t1 = ( "1", );
```

Observação:

A colocação da vírgula é indispensável.

#tupla multiplas (ou empacotamento de sequencia)

```
t2 = ( "1", 2, False );
```

```
t3 = t0, t1, t2;
```

#desempacotamento de sequencia (atribuição multipla)

```
x0, x1, x2 = t2;
```

#inversao de valores

```
x0, x1 = x1, x0;
```

- Conjuntos

Coleções de quaisquer tipos de valores sem elementos repetidos, não ordenadas, separados por vírgulas (indicados por colchetes, ou chaves conforme a versão).

Forma geral:

```
<nome do conjunto> = [ ]  
<nome do conjunto> = set (<lista de valores>)
```

Exemplos:

```
#conjunto vazio  
set0 = [ ];  
set0 = set ( [ ] );
```

```
#conjunto unitario  
set1 = set ([ 1 ]);
```

```
#conjunto formado a partir de uma lista  
letras = [ 'a', 'b', 'a', 'c'];  
set2 = set ( letras );
```

- Dicionários

Coleções de pares (chave e valor), onde uma chave serve para recuperar um valor. As chaves normalmente são valores não repetidos e imutáveis (cadeias de caracteres). Os pares são separados por vírgulas (indicados por chaves).

Forma geral:

```
<nome do dicionário> = { }  
<nome do dicionário> = {<chave1>:<valor1>, <chave2>:<valor2>, ... }
```

Exemplos:

```
#dicionario vazio  
dic0 = { };
```

```
#dicionario com um par  
dic1 = { 'key':0 };
```

```
#dicionario mais pares  
dic2 = { 'key':0, "chave":1 };
```

```
#acesso  
primeira_chave = dic2 [ 'key' ];
```

- Tipos de operadores
 - Aritméticos

| Algoritmo | Python |
|---------------------------|----------|
| \wedge | $**$ |
| * / div mod | * / // % |
| + - | + - |

Observações:

O operador **div** (divisão inteira ou //) é aplicado apenas a operandos inteiros.

Existem formas compactas para incremento e decremento:

| | |
|----------------------|----------------|
| <variável inteira>++ | pós-incremento |
| ++<variável inteira> | pré-incremento |
| <variável inteira>-- | pós-decremento |
| --<variável inteira> | pré-decremento |

- Relacionais

| Algoritmo | Python |
|-----------|-----------|
| < ≤ > ≥ | < <= > >= |
| = ≠ | == != <> |

Observação:

O resultado de uma comparação de dois valores pode ser 0 (falso) ou 1 (verdadeiro).

- Lógicos (bit a bit)

| Algoritmo | Python |
|-------------------------|--------|
| complemento de um | ~ |
| e | & |
| ou-exclusivo | ^ |
| ou | |
| deslocamento à direita | >> |
| deslocamento à esquerda | << |

Observação:

O resultado de uma operação lógica é um valor cujos bits são operados um a um de acordo com a álgebra de proposições.

- Conectivos lógicos

| Algoritmo | Python |
|-----------|------------|
| não | not |
| e | and |
| ou | or |

- Operadores de pertinência a grupos

| Algoritmo | Python |
|--------------|---------------|
| pertence | in |
| não pertence | not in |

- Operadores de identidade

| Algoritmo | Python |
|-----------|---------------|
| = | is |
| ≠ | is not |

Observação:

O resultado de uma operação de comparação de identidade é um valor lógico que será igual a verdadeiro apenas quando os dois operandos forem idênticos (mesmo valor e referência).

- Prioridade de operadores

| Operador | Associação |
|--------------------------|------------|
| () [] | à esquerda |
| not ~ + - (sinal) | à esquerda |
| ** | à direita |
| * / // % | à esquerda |
| + - | à esquerda |
| >> << | à esquerda |
| < <= >= > | à esquerda |
| == != <> | à esquerda |
| & | à esquerda |
| ^ | à esquerda |
| and | à esquerda |
| or | à esquerda |
| ?: | à esquerda |
| += -= *= /= %= | à esquerda |
| >>= <<= &= = ^= | à esquerda |
| , | à esquerda |

- Funções intrínsecas

As regras usadas na formação dos nomes dessas funções intrínsecas são as mesmas utilizadas para os nomes das variáveis.

Exemplo:

```
a = sin ( b )
```

a - nome da variável que receberá o resultado da função;
 sin - função (seno) predefinida do Python ;
 b - nome da variável que vai ser o argumento da função.

| Nome (argumento) | Tipo de argumento | Descrição |
|------------------|-------------------|------------------------|
| sin (X) | real | seno (em radianos) |
| cos (X) | real | cosseno (em radianos) |
| atan(X) | real | arco tangente |
| sqrt(X) | real | raiz quadrada |
| exp (X) | real | exponencial de "e" |
| abs (X) | int | valor absoluto inteiro |
| fabs(X) | real | valor absoluto real |
| log (X) | real | logaritmo neperiano |
| log10 (X) | real | logaritmo neperiano |
| pow(X,Y) | real, real | eleva X a Y |

A linguagem Python dispõe de uma significativa biblioteca básica, com diversas funções além das aritméticas citadas acima.

- Expressões

- Aritmética

Exemplos:

| Algoritmo | Python |
|-----------------|---------------------|
| 10 + 15 | 10 + 15 |
| 543.15/3 | 543.15/3 |
| (x + y + z)*a/z | ((x + y + z) * a)/z |

- Lógica

Exemplos:

| Algoritmo | Python |
|-------------------|----------------------|
| A = 0 | A == 0 |
| a ≠ 1 | a != 1 |
| (A ≥ 0) & (a ≤ 1) | (A >= 0) && (a <= 1) |

Observação:

Para efeito de clareza, ou para mudar a precedência de operadores, pode-se separar as proposições por espaços e agrupá-las com parênteses.

- Estrutura de programa

```
# definicoes para pre-processamento
import sys
from math import *
from random import random

# definicoes globais

# definicoes de modulos, funcoes e procedimentos
def <nome do modulo> ( <lista de parametros> ):
    # definicoes locais

    # comandos
    return

# parte principal
# definicoes locais

# comandos
```

- Comentários

Comentários são precedidos pelos sinais # , para uma única linha
"""...""" envolvendo todo o texto, distribuído em múltiplas linhas.

Exemplo:

```
# exemplo de programa

# Este programa nao faz nada - comentario

"""
    que também
    pode ser colocado
    assim
"""
```

- Atribuição

- Atribuição simples

Forma geral:

<variável> = <expressão>;

Exemplo:

```
x    = 0 ;  
a    = 1.57;  
letra = 'A' ;
```

- Atribuição múltipla

Forma geral:

<variável 1> = <variável 2> = <expressão>;

Exemplo:

```
x = y = 0;
```

Observação:

A execução inicia-se pela direita.

- Atribuição composta

Forma geral:

<variável> <operador> = <expressão>;

Exemplo:

```
i += 1   ou   i = i + 1
```

Observação:

Operadores permitidos: ** + - * / % >> << | & ^

- Atribuição condicional

Forma geral:

<variável> = <teste> ? <expressão 1>: <expressão 2>;

Exemplo:

```
x = (a < b) ? a: b;
```

- Descrição de entrada e saída
- Entrada/Saída (padrão Python):

Forma geral:

```
<variável> = input ( "mensagem" );

print < expressão>;
printf <formato> % <lista de itens>;
```

- Especificação de formatos:

Forma geral:

```
%<sinais><<0><largura>>< . ><precisão><conversão>
```

onde:

| | |
|-------------|------------------------------------------------------------------------------------------------------------|
| <sinais> | podem ser: |
| '.' | - alinha a esquerda a saída |
| '+' | - conversão de sinal (+ ou -) |
| ' ' | - conversão de sinal (" " ou -) |
| '#' | - começo com (0, 0x onde apropriado) |
| <0> | - preenchimento com zeros |
| <largura> | - largura mínima do campo |
| <precisão> | - número máximo de caracteres - ou número de dígitos fracionários (neste caso é precedida por < . >) |
| <conversão> | - pode ser: |

| caractere | argumento | conversão |
|-----------|------------|---------------------------|
| d | inteiro | para decimal |
| o | inteiro | para octal |
| x | inteiro | para hexadecimal |
| u | inteiro | para decimal, sem sinal |
| c | caractere | para um caractere |
| s | caracteres | para cadeia de caracteres |
| e | real | para real com expoente |
| f | real | para real sem expoente |
| g | real | para real |
| % | | sinal % |
| ld | inteiro | para decimal |
| lo | inteiro | para octal |
| lx | inteiro | para hexadecimal |

Exemplos:

%5c - X do tipo caractere e com valor igual a 'A'

| | | | | |
|--|--|--|--|---|
| | | | | A |
|--|--|--|--|---|

%5d - X do tipo inteiro e com valor igual a 100

| | | | | |
|--|--|---|---|---|
| | | 1 | 0 | 0 |
|--|--|---|---|---|

%5.2f - X do tipo real e com valor igual a -1

| | | | | |
|---|---|---|---|---|
| - | 1 | . | 0 | 0 |
|---|---|---|---|---|

Observação:

Se a largura (no exemplo, 5) não for suficiente para conter o número na sua forma de representação interna, o tamanho padrão para cada tipo será usado.

- Caracteres com funções especiais em formatos:

| caractere | função |
|-----------|------------------------------|
| \0 | fim da cadeia de caracteres |
| \n | fim de linha (LF) |
| \t | tabulação |
| \b | retrocesso (BS) |
| \r | retorno de carro (CR) |
| \f | avanço de carro (FF) |
| \\ | barra invertida |
| \' | apóstrofo |
| \onn | representação em octal |
| \xnn | representação em hexadecimal |

Exemplo completo de programa:

```
# dependencias
import sys
from math import *
from random import random

# definicao de modulo
def readint ( parameter ):
    x = input ( parameter )
    return x

# parte principal
i = 0; j = 0;

print "Exemplo: ";
print "\n";
j = readint ( "Digitar um numero inteiro: " );
i = j * 2 + 10;
print "%s %d\n" % ( "O resultado e' igual a ", i );
raw_input ( "\nPRESSIONAR <Enter> para terminar." );
```

Se fornecido o valor 5 para a variável *j* , o resultado será:

O resultado e' igual a 20

- Estruturas de controle
- Sequência simples

Forma geral:

| Algoritmo | Python |
|-----------|-----------------------|
| <comando> | <comando> |
| <comando> | <comando> ; <comando> |

Observação:

Em Python normalmente há um comando por linha, dispensando-se o ponto-e-vírgula. Se houve mais de um comando por linha, deverão ser separados por ponto-e-vírgula. É recomendável usar endentação para definir blocos de comandos.

- Estrutura alternativa
- Alternativa simples

Forma geral:

| Algoritmo | Python |
|---------------|------------------------|
| se <condição> | if (<condição>) |
| então | |
| <comandos> | <comandos> ; |
| fim se | |

Observação:

Se houver outra alternativa interna a essa, usar endentação é obrigatório.

Exemplo:

```

if ( x >= 0 )
    <comando1, da primeira alternativa>;
    if ( x == 0 )
        <comando2, da segunda alternativa>;
        <comando3, da segunda alternativa>
    <comando4, ainda da primeira alternativa>;
<outro comando, fora da alternativa>

```

- Alternativa dupla

Forma geral:

| Algoritmo | Python |
|---------------------|-------------------------|
| se <condição> então | if (<condição>): |
| <comandos 1> | <comandos 1> ; |
| senão | else: |
| <comandos 2> | <comandos 2> ; |
| fim se | |

Observação:

Se houver outra alternativa interna a essa, usar endentação é obrigatório.

Exemplo:

```

if ( x >= 0 ):
    <comando1, da primeira alternativa>;
    if ( x == 0 ):
        <comando2, da segunda alternativa>;
        <comando3, da segunda alternativa>
    else:
        <comando4, da segunda alternativa>
    <comando6, ainda da primeira alternativa>;
else:
    <comando7, da primeira alternativa>;
    <comando8, da primeira alternativa>
<outro comando, fora da alternativa>

```

- Alternativa múltipla

Forma geral:

| Algoritmo | Python |
|------------------|----------------------------------------|
| escolher <valor> | if <valor> == <opção 1>: |
| <opção 1>: | <comandos 1> ; |
| <comandos 1> | |
| <opção 2>: | elif <valor> == <opção2>: |
| <comandos 2> | <comandos 2> ; |
| ... | ... |
| <opção n-1>: | elif <valor> == <última opção>: |
| <comandos N-1> | <comandos N-1> ; |
| senão | else: |
| <comandos N> | <comandos N> |
| fim escolher | |

Observação:

A indicação **else** é opcional.

- Estrutura repetitiva

- Repetição com teste no início

Forma geral:

| Algoritmo | Python |
|-----------------------------|---------------------------|
| repetir enquanto <condição> | while <condição> : |
| <comandos> | <comandos> ; |
| fim repetir | |

Observação:

A condição para execução é sempre verdadeira.

- Repetição com teste no início e variação

Forma geral:

| Algoritmo | Python |
|------------------------------|------------------------------------------|
| repetir para | for <variável> in xrange (|
| <variável> = <valor inicial> | <valor inicial> , |
| : <fim> | <teste de fim> , |
| : <variação> | <variação>) |
| <comandos> | <comandos> ; |
| fim repetir | |

Observações:

A condição para execução é sempre verdadeira.

Em Python , qualquer um dos elementos, ou mesmo todos, podem ser omitidos. Entretanto, se tal for preciso, recomenda-se o uso de outra estrutura mais apropriada.

- Repetição com teste no fim

Forma geral:

| Algoritmo | Python |
|------------------------|---------------------------|
| repetir até <condição> | condition = True ; |
| <comandos> | while condition : |
| fim repetir | <comandos> ; |
| | condition = <expressão> ; |

- Interrupções

Em Python , as repetições podem ser interrompidas, em sua sequência normal de execução através dos comandos:

break; e **continue;**

O comando **break** serve para interromper completamente uma repetição, passando o controle ao próximo comando após a estrutura repetitiva.

O comando **continue** interrompe uma iteração, voltando ao início.