Python:
Tipos Básicos

Claudio Esperança

Python como calculadora

- O Interpretador python pode ser usado como calculadora
- Por exemplo, as quatro operações aritméticas são denotadas pelos símbolos
  - + adição
  - - subtração
  - * multiplicação
  - / divisão

Python como calculadora

```
>>> 10
10
>>> # Um comentário é precedido do caracter "#"
... # Comentários são ignorados pelo interpretador
... 10+5
15
>>> 10-15 # Comentários podem aparecer também após código
-5
>>> 10*3
30
>>> 10/3
3
>>> 10/-3 # Divisão inteira retorna o piso
-4
>>> 10%3 # Resto de divisão inteira simbolizado por %
1
```

Tipos de dados

- São categorias de valores que são processados de forma semelhante
- Por exemplo, números inteiros são processados de forma diferente dos números de ponto flutuante (decimais) e dos números complexos
- Tipos primitivos: são aqueles já embutidos no núcleo da linguagem
  - Simples: números (int, long, float, complex) e cadeias de caracteres (strings)
  - Compostos: listas, dicionários, tuplas e conjuntos
- Tipos definidos pelo usuário: são correspondentes a classes (orientação objeto)

Variáveis

- São nomes dados a áreas de memória
  - Nomes podem ser compostos de algarismos,letras ou _
  - O primeiro caractere não pode ser um algarismo
  - Palavras reservadas (`if`, `while`, etc) são proibidas
- Servem para:
  - Guardar valores intermediários
  - Construir estruturas de dados
- Uma variável é modificada usando o comando de atribuição:

  Var = expressão
- É possível também atribuir a várias variáveis simultaneamente:

  var1,var2,...,varN = expr1,expr2,...,exprN

Variáveis

```
>>> a=1
>>> a
1
>>> a=2*a
>>> a
2
>>> a,b=3*a,a
>>> a,b
(6, 2)
>>> a,b=b,a
>>> a,b
(2, 6)
```

Variáveis

- Variáveis são criadas dinamicamente e destruídas quando não mais necessárias, por exemplo, quando saem fora de escopo (veremos isso mais tarde)
- O *tipo* de uma variável muda conforme o valor atribuído, i.e., int, float, string, etc.
  - Não confundir com linguagens *sem tipo*
  - Ex.:

```
>>> a ="1"
>>> b = 1
>>> a+b
Traceback (most recent call last):
  File "<stdin>", line 1, in ?
TypeError: cannot concatenate 'str' and 'int' objects
```

Números

- Há vários tipos numéricos que se pode usar em python
  - **Int**: números inteiros de *precisão fixa*
    - 1 , 2 , 15 , -19
  - **Long**: números inteiros de *precisão arbitrária*
    - 1L , 10000L , -9999999L
  - **Floats**: números racionais de precisão *variável*
    - 1.0 , 10.5 , -19000.00005 , 15e-5
  - **Complex**: números complexos
    - 1+1j , 20j , 1000+100J

Números inteiros

- Os **int**s têm precisão fixa ocupando tipicamente uma palavra de memória
  - Em PC's são tipicamente representados com 32 bits (de $-2^{31}-1$ a 2^{32})
- Os números inteiros de precisão arbitrária (**long**s) são armazenados em tantas palavras quanto necessário
  - Constantes do tipo long têm o sufixo L ou l
  - Longs são manipulados bem mais lentamente que ints
  - Quando necessário, cálculos usando ints são convertidos para longs

Números inteiros

```
>>> a=2**30 # Potenciação
>>> a
1073741824
>>> b=a*1000
>>> b
1073741824000L
>>> b/1000
1073741824L
```

Números inteiros

- Constantes podem ser escritas com notação idêntica à usada em C
  - Hexadecimal: preceder dígitos de 0x
  - Octal: preceder dígitos de 0
  - Ex.:

```
>>> 022
18
>>> 0x10
16
>>> 0x1f
31
```

Números de ponto flutuante

- São implementados como os **double**'s da linguagem C – tipicamente usam 2 palavras
- Constantes têm que possuir um ponto decimal ou serem escritas em notação científica com a letra “e” (ou “E”) precedendo a potência de 10
- Ex:

```
>>> 10 # inteiro
10
>>> 10.0 # ponto flutuante
10.0
>>> 99e3
99000.0
>>> 99e-3
0.09900000000000005
```

Números complexos

- Representados com dois números de ponto flutuante: um para a parte real e outro para a parte imaginária
- Constantes são escritas como uma soma sendo que a parte imaginária tem o sufixo j ou J
- Ex.:

```
>>> 1+2j
(1+2j)
>>> 1+2j*3
(1+6j)
>>> (1+2j)*3
(3+6j)
>>> (1+2j)*3j
(-6+3j)
```

Strings

- São cadeias de caracteres
- Constituem outro tipo fundamental do python
- Constantes ***string*** são escritas usando aspas simples ou duplas
  - Ex.: `"a" ou 'a'`
- O operador "+" pode ser usado para concatenar strings
  - Ex.: `"a"+"b"` é o mesmo que `"ab"`
- O operador "*" pode ser usado para repetir strings
  - Ex.: `"a"*10` é o mesmo que `"aaaaaaaaaa"`

Strings

- Python usa a tabela de caracteres default do S.O.
  - Ex.: ASCII, UTF-8
- Caracteres não imprimíveis podem ser expressos usando notação "barra-invertida" (\)
  - \n é o mesmo que *new line*
  - \r é o mesmo que *carriage return*
  - \t é o mesmo que *tab*
  - \b é o mesmo que *backspace*
  - \\ é o mesmo que \
  - \x41 é o mesmo que o caractere cujo código hexadecimal é 41 ("A" maiúsculo)

Strings

```
>>> "ab\rd"
'ab\rd'
>>> print "ab\rd" # print exibe chars não imprimíveis
db
>>> print "abc\td"
abc     d
>>> print "abc\nd"
abc
d
>>> print "abc\\nd"
abc\nd
>>> print "ab\bc"
ac
>>> print "\x41\xA1"
Aí
```

Strings

- A notação *barra-invertida* (\) pode ser desabilitada desde que a constante string seja precedida por um r (erre minúsculo)
  - São chamadas strings *raw* (cruas)
  - Ex.:

```
>>> print "abc\ncd\tef"
abc
cd      ef
>>> print r"abc\ncd\tef"
abc\ncd\tef
```

Strings

- Constantes string podem ser escritas com várias linhas desde que as aspas não sejam fechadas e que cada linha termine com uma barra invertida
- Ex.:

```
>>> print "abcd\n\
... efgh\n\
... ijk"
abcd
efgh
ijk
>>> print "abcd\
... efgh\
... ijk"
abcdefghijk
>>>
```

Strings

- Também é possível escrever constantes string em várias linhas incluindo as quebras de linha usando três aspas como delimitadores
- Ex.:

```
>>> print """
Um tigre
dois tigres
três tigres"""

Um tigre
dois tigres
três tigres
>>> print '''abcd
efgh'''
abcd
efgh
```

Strings – Índices

- Endereçam caracteres individuais de uma string
  - Notação: *string*[*índice*]
  - O primeiro caractere tem índice 0
  - O último caractere tem índice -1
  - Ex.:

```
>>> a = "abcde"
>>> a[0]
'a'
>>> a[-1]
'e'
```

Strings – Fatias (slices)

- Notação para separar trechos de uma string
  - Notação: *string*[*índice1*:*índice2*]
  - Retorna os caracteres desde o de índice1 (inclusive) até o de índice2 (exclusive)
  - Se o primeiro índice é omitido, é assumido 0
  - Se o último índice é omitido, é assumido o fim da string

Strings – Fatias (slices)

```
>>> a
'abcde'
>>> a[0:2]
'ab'
>>> a [2:]
'cde'
>>> a[:]
'abcde'
>>> a[-1:]
'e'
>>> a[:-1]
'abcd'
```

Expressões booleanas

- Também chamadas expressões lógicas
- Resultam em verdadeiro (True) ou falso (False)
- São usadas em comandos condicionais e de repetição
- Servem para analisar o estado de uma computação e permitir escolher o próximo passo
- Operadores mais usados
  - Relacionais: > , < , ==, !=, >=, <=
  - Booleanos: and, or, not
- Avaliação feita em “Curto-circuito”
  - Expressão avaliada da esquerda para a direita
  - Se o resultado (verdadeiro ou falso) puder ser determinado sem avaliar o restante, este é retornado imediatamente

Expressões booleanas

```
>>> 1==1
True
>>> 1==2
False
>>> 1==1 or 1==2
True
>>> 1==1 and 1==2
False
>>> 1<2 and 2<3
True
>>> not 1<2
False
>>> not 1<2 or 2<3
True
>>> not (1<2 or 2<3)
False
>>> "alo" and 1
1
>>> "alo" or 1
'alo'
```

Expressões booleanas

- As constantes **`True`** e **`False`** são apenas símbolos convenientes
- Qualquer valor não nulo é visto como verdadeiro enquanto que **`0`** (ou **`False`**) é visto como falso
- O operador **`or`** retorna o primeiro operando se for vista como *verdadeiro*, caso contrário retorna o segundo
- O operador **`and`** retorna o primeiro operando se for vista como *falso*, caso contrário retorna o segundo
- Operadores relacionais são avaliados antes de **`not`**, que é avaliado antes de **`and`**, que é avaliado antes de **`or`**

Expressões booleanas

```
>>> 0 or 100
100
>>> False or 100
100
>>> "abc" or 1
'abc'
>>> 1 and 2
2
>>> 0 and 3
0
>>> False and 3
False
>>> 1 and 2 or 3
2
>>> 0 or 2 and 3
3
>>> 1 and not 0
True
```

Funções Embutidas

- Além dos operadores, é possível usar funções para computar valores
- As funções podem ser definidas:
  - Pelo programador (veremos + tarde)
  - Em módulos da biblioteca padrão
  - Por *default:* são as funções *embutidas* (*built-in*)
    - Na verdade, fazem parte do módulo __builtins__, que é sempre importado em toda aplicação
- Ex.:
  - abs(*x*) retorna o valor absoluto do número *x*
  - chr(*x*) retorna uma string com um único caractere cujo código ASCII é *x*
  - ord(*s*) retorna o código ASCII do caractere *s*

Funções Embutidas

```
>>> abs (10)
10
>>> abs (-19)
19
>>> chr (95)
'_'
>>> chr (99)
'c'
>>> ord ('a')
97
```

Importando módulos

- Muitas funções importantes são disponibilizadas em módulos da biblioteca padrão
  - Ex.: o módulo **`math`** tem funções transcendentais como **`sin`**, **`cos`**, **`exp`** e outras
- Um módulo pode conter não só funções mas também variáveis ou classes
  - Por exemplo, o módulo **`math`** define a constante **`pi`**
- Para usar os elementos de um módulo, pode-se usar o comando **`import`**
  - Formatos:
    - **`import`** *`modulo`*
    - **`from`** `modulo` **`import`** *`nome,...,nome`*
    - **`from`** `modulo` **`import`** `*`

Importando módulos

- Por exemplo:
  - `from math import *`
    `# importa todos os elementos do módulo math`
  - `from math import sin`
    `# importa apenas a função sin`
  - `import math`
    `# importa o módulo math como um todo`
    `# (todos os elementos têm que ser citados`
    `# precedidos por` **`math.`**`)`

Importando módulos

```
>>> import math
>>> a = sin(30)
Traceback (most recent call last):
  File "<stdin>", line 1, in ?
NameError: name 'sin' is not defined
>>> a = math.sin(30)
>>> from math import sin
>>> a = sin(30)
>>> print a
-0.988031624093
>>> a = sin(radians(30))
Traceback (most recent call last):
  File "<stdin>", line 1, in ?
NameError: name 'radians' is not defined
>>> from math import *
>>> a = sin(radians(30))
>>> a
0.49999999999999994
```

Explorando Módulos

```
>>> import math
>>> help(math.cos)
Help on built-in function cos in module math:

cos(...)
    cos(x)

    Return the cosine of x (measured in radians).
(END)
```

- Pressiona-se “q” para retornar ao interpretador.