SCC 124 - Introdução à Programação para Engenharias

Tipos de Dados e Variáveis

Professor: André C. P. L. F. de Carvalho, ICMC-USP
Pos-doutorando: Isvani Frias-Blanco
Monitor: Henrique Bonini de Britto Menezes

Aula de hoje

- Dados
- Constantes literais
- Variáveis
- Tipos de dados
- Números
- *Strings*

Dados

- Uma das principais características dos programas é que eles manipulam dados
  - Programas precisam armazenar dados
    - Valores dos dados não mudam: constantes literais
    - Valores podem mudar: variáveis
      - Cada variável armazena valores de um tipo de dado
  - Existem vários tipos de dados diferentes
    - Números
    - Textos
    - Tipos mais complexos

Constantes literais

- Valor é usado literalmente (não são variáveis)
- Facilita uso e manutenção de um valor
- Exemplos
  - *7*
  - *"texto constante"*
  - *4.765*
- Ao contrário de outras linguagens, Python não tem declaração de constantes
  - Na linguagem C, *#define minimo 0*

Tipagem dinâmica

- Usada por Python
- Valor da variável define seu tipo
- Tipo da variável pode mudar durante execução de um programa
  - Uma variável pode armazenar um valor inteiro em um ponto de um programa
    - E armazenar um *string* mais adiante

Variáveis

- Sintaxe para declarar variáveis em Python:
  - *Nome = valor*
- Propriedades de variáveis:
  - Nome
  - Tipo
  - Tempo de vida
  - Escopo

Convenções para nomes de variáveis

- Deve iniciar com letra ou *underscore* (_)
  - Demais caracteres de um nome devem ser letras, números ou *underscores*
  - MAIÚSCULAS ≠ minúsculas
  - Nomes longos dificultam leitura de expressões
- Bons nomes reduzem necessidade de comentários
- Não utilizar palavras-chave como nomes
- Também valem para funções

Palavras-chave (*keywords*)

and	*as*	*assert*	*break*	*class*
continue	*def*	*del*	*elif*	*else*
except	*False*	*finally*	*for*	*from*
global	*if*	*import*	*in*	*is*
lambda	*nonlocal*	*None*	*not*	*or*
pass	*raise*	*return*	*True*	*try*
while	*with*	*yield*		

Palavras-chave (*keywords*)

- Biblioteca
- Não precisa memorizar quais são
- *keyword.iskeyword(aaa)*
  - Retorna se *aaa* é uma palavra chave de Python
- *keyword.kwlist*
  - Retorna todas as palavras chave de Python

Convenções para nomes de variáveis

- Estilo de programação pode ser melhorado adotando convenções
  - Nomes de variáveis e tipos de dados começam com letras minúsculas
  - Nomes de funções começam com letras MAIÚSCULAS

Escopo

- Define área de atuação da variável
  - Fora do escopo, variável não existe
- Pode ser:
  - Local
    - Variáveis locais
  - Global
    - Variáveis globais

Variáveis locais

- Em geral declaradas no início de uma função
  - Mas também podem ser declaradas depois, dentro da função
- Escopo é a função onde elas aparecem
  - Outras funções não têm acesso direto a elas
    - Exceto as funções internas a essa função

Variáveis globais

- Aparecem fora de qualquer definição de função
  - Escopo é o resto do arquivo onde são declaradas
- Podem ser acessadas por qualquer função
  - Permite que funções possam interagir umas com as outras
- **Cuidado**: fácil usar em excesso
  - **Não podem ser utilizadas neste curso**

Tempo de vida

- Variáveis locais
  - Enquanto a função em que foi declarada estiver ativa
    - Chamada da função aloca espaço para as variáveis
    - Término de execução da função libera o espaço alocado
- Variáveis globais
  - Tempo de execução do programa

Tipos de dados

- Principais tipos primitivos (pré-definidos)
  - Numérico ou escalar
    - Inteiro, ponto flutuante (real), números complexos
  - Sequência
    - *Strings* (cadeias de caracteres), listas e tuplas
  - Mapeamento (Dicionário)
  - Classe
  - Instância
  - Exceção
  - Função

Tipos de dados

- Definidos pelo programador
  - Classes (orientação a objetos)
- Tudo em Python é um objeto
  - Incluindo números, *strings* e funções

Tipos numéricos

- Semelhantes aos tipos numéricos de outras linguagens
  - Inteiro (*int*)
  - Ponto flutuante (*float*)
  - Número complexo (*complex*)

Tipo inteiro

- Inteiro normal (*int*)
  - Precisão ilimitada
    - Pode utilizar tantos bits quanto necessário
      - Ao contrário da maioria das linguagens
  - Tipo booleano (*bool*)
    - Subtipo de inteiro
    - Adicionado a Python na versão 2.3
      - Versões anteriores não usavam um tipo booleano especial, mas os valores 1 e 0

Tipo inteiro

- Bases numéricas
  - Valores inteiros geralmente são usados na base decimal
  - Para bases octal e hexadecimal
    - Base binária: começar o número com 0b (zero b)
      - 0b1001= 9 (decimal)
    - Base octal: começar o número com 0o (zero o)
      - 0o40 = 32 (decimal)
    - Base hexadecimal: começar o número com 0x (zero x)
      - 0xFF = 255 (decimal)

Tipo inteiro

- Valores binários
  - Números hexadecimais são geralmente usados para descrever valores binários
    - 2 dígitos hexadecimais correspondem a exatamente 1 byte

Exemplo

```
>>> 2+2
4
>>> # Isso é um comentário
>>> 2+2 #Agora um comentário na mesma linha
4
>>> (50-5*6)/4
5
>>> # Divisão por inteiro arredonda resultado:
>>> 7/3
2.333333
>>> 7/-3
-2.3333
```


Exercício

- O que faz o código a seguir?

```
print (" %s %s %s %s" % ('%6d', '%06d', '%6x' , '%6o'))
for valor in range (16):
    print ("%6d %06d %6x %6o" % (valor, valor, valor, valor))
    print ()
```


Exercício

```
%6d %06d %6x %6o
     0 000000      0      0
     1 000001      1      1
     2 000002      2      2
     3 000003      3      3
     4 000004      4      4
     5 000005      5      5
     6 000006      6      6
     7 000007      7      7
     8 000008      8     10
     9 000009      9     11
    10 000010      a     12
    11 000011      b     13
    12 000012      c     14
    13 000013      d     15
    14 000014      e     16
    15 000015      f     17

Process finished with exit code 0
```


Tipo Booleano

- Contém apenas dois valores:
  - *True* (verdadeiro)
  - *False* (falso)
- Equivale aos valores 0 (*False*) e 1 (*True*)
  - X == 0 e x != 0
- Podem ser combinados com valores numéricos em expressões aritméticas
- Conversão para *strings* retorna os valores "*True*" ou "*False*"

Exemplo

```
>>> bool(1)
True
>>> bool(0)
False
>>> bool([])
False
>>> bool (-21)
True
>>> bool( (1,) )
True
```

```
>>> True + 1
2
>>> False + 1
1
>>> False * 9
0
>>> (True+True) * 7
14
```


Tipo ponto flutuante

- Números reais em ponto flutuante (*float*)
- Aumenta precisão, em relação a *int*, às custas de mais espaço de memória
- Utiliza mesmas regras para escrever valores de ponto flutuante (*double*) da linguagem C
  - Ex. 4.01, 2.34e+9, 4E120
    - Notação E indica potência de 10
      - 4.7E-3 = 4.7 * 10^{-3}

Tipo números complexos

- Escritos na forma *5 + 7j* ou *complex (5, 7)*
  - Parte imaginária (termina com j ou J)
  - Parte real
- Parte real é opcional
  - Parte imaginária pode vir no início
- Implementado como um par de números do tipo *float*
  - Operações numéricas para números complexos são usadas

Exemplo

```
>>> 1j * 1j  # imaginários
(-1+0j)
>>> 1j * complex(0,1)
(-1+0j)
>>> 3+1j*3
(3+3j)
>>> (3+1j)*3
(9+3j)
>>> (1+2j)/(1+1j)
(1.5+0.5j)
```

```
>>> (3+4j).real
3.0
>>> (3+4j).imag
4.0
>>> a =1.5+0.5j
>>> a.real
1.5
>>> a.imag
0.5
```


Tipo *string*

- Coleção ordenada de caracteres
  - Ordem pré-definida de caracteres, da esquerda para a direita
  - E.: "Engenharia"
- Subtipo do tipo *sequence*
  - Inclui também listas e tuplas

Tipo *string*

- Usado para armazenar e representar sequências de caracteres
- Pode representar
  - Nomes
  - Frases
  - Sequências de DNA
- Python possui vários operadores para manipular *strings*

Tipo *string*

- Um *string* pode ser encaixado entre aspas simples ou duplas
  - Ex.: "*Tudo bem?*" = ´*Tudo bem?*´
  - Usa-se aspas triplas, ''' ou """, para *strings* de várias linhas
    - Aspas duplas ou simples podem ser usadas dentro de um *string*

```
'''This is a multi-line string. This is the first line.
This is the second line.
"What's your name?," I asked.
He said "Bond, James Bond."
'''
```


Método *format*

- Permite construir *strings* usando informação adicional
  - Ex.: seja o código armazenado no arquivo texto.py

```
idade = 20
nome = 'Asdrubal'
print ('{0} tinha {1} anos quando entrou na USP'.format(nome, idade))
print ('Por que {0} esta cursando ICC?'.format(nome) )
```

Imprime:

```
$ python texto.py
Asdrubal tinha 20 anos quando entrou na USP
Por que Asdrubal esta cursando ICC?'
```


Método *format*

- Alternativas:

```
idade = 20
nome = 'Asdrubal'
nome + 'tinha' + str(idade) + 'anos' + 'quando entrou na USP'
print ('Por que'+ nome + 'esta cursando IPE?')
```

```
idade = 20
nome = 'Asdrubal'
print ('{ } tinha { } anos quando entrou na USP'.format(nome, idade))
print ('Por que { } esta cursando IPE?'.format(nome) )
```


Método *format*

- Qual a saída do código abaixo?

```
# decimal (.) precisão 3 para float '0.333'
print ('{0:.3f}'.format(1.0/3) )
# preenche com underscores (_) texto centralizado
# (^) para largura 11 '____ola____'
print ('{0:_^11}'.format('ola') )
# texto 'Asdrubal passou na USP'
print ('{nome} passou na {uni}'.format(nome='Ana', uni='USP'))
print ('{} passou na {}'.format('Ana', 'USP'))
```


Método *format*

- Qual a saída do código abaixo?

```
# decimal (.) precisao 3 para float '0.333'
print ('{0:.3f}'.format(1.0/3) )
# preenche com underscores (_) texto centralizado
# (^) para largura 11 '____ola____'
print ('{0:_^11}'.format('ola') )
# texto 'Asdrubal passou na USP'
print ('{nome} passou na {uni}'.format(nome='Ana', uni='USP'))
print ('{} passou na {}'.format('Ana', 'USP'))
```

```
0.333
____ola____
Ana passou na USP
Ana passou na USP
```


Exemplos de uso de *strings*

```
>>> 'Quer um suco de laranja?'
'Quer um suco de laranja?'
>>> 'e um copo d\'agua?'
"e um copo d'agua ?"
>>> "e um copo d'agua?"
"e um copo d'agua ?"
>>> "'Sim,' por favor."
"'Sim,' por favor."
>>> "'Sim,\' por favor."
"'Sim,' por favor."
```

Exemplos de uso de *strings*

```
>>> dna1 = 'gcatgac'
>>> dna1
'gcatgac'
>>> dna2 = 'cctttact'
>>> dna1 = dna1 + dna2
>>> dna1
'gcatgaccctttact'
```


Conclusão

- Dados
- Constantes literais
- Variáveis
- Tipos de dados
- Números
- *Strings*

Perguntas