Algoritmos e Estrutura de Dados

Aula 1

Introdução a Linguagem Python (Parte I)

Prof. Tiago A. E. Ferreira

Linguagem a ser Utilizada?

- Nossa disciplina é de Algoritmos e Estrutura de Dados, e não de linguagem de programação
  - Porém, é fundamental a definição de uma linguagem de programação para padronização da disciplina
- A maioria esmagadora das referências para estrutura de dados utilizam:
  1. C/C++
  2. Java
- A vocação do DEINFO/UFRPE:
  - A linguagem Python !!!!!

A Linguagem Python

- A linguagem Python
  - Desenvolvida no final dos anos de 1980
  - Nome inspirado no *Monty Python's Flying Circus*
  - É uma linguagem orientada a objeto
- Vantagens de Python
  - É uma linguagem versátil e muito elegante
  - É uma linguagem de rápida prototipação, quando comparada a C/C++ ou Fortran.
- Desvantagens de Python
  - É uma linguagem interpretada.
    - Não é capaz de gerar uma aplicação que rode *stand-alone*
    - Para se rodar um programa Python é necessário ter o interpretador instalado!

Outras vantagens...

- Python é um software *open-source.*
  - *Sendo residente dos sistemas linux.*
- Python está disponível para todos os grandes sistemas.
  - Linux, windows, Mac/OS, etc.
- Python é portável.
- O código Python é intuitívo e bastante amigável.
- O Python, suas extenções e bibliotecas são de facil instalação.

Onde Encontrar o Python

- É possível obter o interpretador Python no site:
  - www.python.org
  - Há também o site do *pythonBrasil:*
    - www.python.org.br
  - Listas de discussão sobre Python:
    - No Brasil
      - http://www.python.org.br/wiki/EnvolvaSe
    - No Mundo
      - http://mail.python.org/mailman/listinfo

Variáveis em Python

- Numéricas:

```
>>> b = 2          # b is integer type
>>> print b
2
>>> b = b * 2.0  # Now b is float type
>>> print b
4.0
```

- Strings:

```
>>> string1 = 'Press return to exit'
>>> string2 = 'the program'
>>> print string1 + ' ' + string2  # Concatenation
Press return to exit the program
>>> print string1[0:12]            # Slicing
Press return
```

  - Uma string é um objeto imutável:

```
>>> s = 'Press return to exit'
>>> s[0] = 'p'
Traceback (most recent call last):
  File ''<pyshell#1>'', line 1, in ?
    s[0] = 'p'
TypeError: object doesn't support item assignment
```

Exemplo: Python como uma Calculadora

```
>>> 2+2
4
>>> _ + 7
11
>>> (50-5*5)/4
6
>>> largura, altura = 13.4, 20.2
>>> largura * altura
270.68000000000001
>>> (1+2j)/(1+1j)
(1.5+0.5j)
>>> a=1.5+0.5j
>>> a.real
1.5
>>> a.imag
0.5
```

Exemplo: Algumas Operações Básicas com Strings

```
>>> "strings podem estar entre aspas duplas"
'strings podem estar entre aspas duplas'
>>> 'ou entao entre as simples!'
'ou entao entre as simples!'
>>> a = 'Jovens'
>>> a + ' sao rebeldes'
'Jovens sao rebeldes'
>>> 5*a
'JovensJovensJovensJovensJovens'
>>> a[0]
'J'
>>> a[1:3]
'ov'
>>> a[2:]
'vens'
```

Tuplas

- Uma **tupla** é uma sequência qualquer de objetos separados por vírgulas, contidos entre parênteses.

```
>>> rec = ('Smith','John',(6,23,68))    # This is a tuple
>>> lastName,firstName,birthdate = rec  # Unpacking the tuple
>>> print firstName
John
>>> birthYear = birthdate[2]
>>> print birthYear
68
>>> name = rec[1] + ' ' + rec[0]
>>> print name
John Smith
>>> print rec[0:2]
('Smith', 'John')
```

Lista

- Um **Lista** em Python é semelhante as tuplas, sendo *mutáveis* e o agrupamento é feito com colchetes.

```
>>> a = [1.0, 2.0, 3.0]      # Create a list
>>> a.append(4.0)            # Append 4.0 to list
>>> print a
[1.0, 2.0, 3.0, 4.0]
>>> a.insert(0,0.0)          # Insert 0.0 in position 0
>>> print a
[0.0, 1.0, 2.0, 3.0, 4.0]
>>> print len(a)             # Determine length of list
5
>>> a[2:4] = [1.0, 1.0] # Modify selected elements
>>> print a
[0.0, 1.0, 1.0, 1.0, 1.0, 4.0]
```

Copiando Objetos Mutáveis

- Se **a** é uma objeto mutável, por exemplo uma **lista**, se fizermos ***b = a***, o objeto **b** é uma **referência do objeto a**.
  - Assim qualquer alteração em **b** irá refletir em **a**.
- Para realizar uma cópia independente, faça:
  - **b = a[:]**

```
>>> a = [1.0, 2.0, 3.0]
>>> b = a                 # 'b' is an alias of 'a'
>>> b[0] = 5.0            # Change 'b'
>>> print a
[5.0, 2.0, 3.0]           # The change is reflected in 'a'
>>> c = a[:]              # 'c' is an independent copy of 'a'
>>> c[0] = 1.0            # Change 'c'
>>> print a
[5.0, 2.0, 3.0]           # 'a' is not affected by the change
```

Construindo Matrizes

- Em Python é possível construir matrizes através de **listas**

```
>>> a = [[1, 2, 3], \
         [4, 5, 6], \
         [7, 8, 9]]
>>> print a[1]           # Print second row (element 1)
[4, 5, 6]
>>> print a[1][2]        # Print third element of second row
6
```

Operadores

- Aritméticos:

+	Adição
-	Subtração
*	Multiplicação
/	Divisão
**	Exponenciação
%	Divisão modular (mod)

- Operações:

a += b	a = a + b
a -= b	a = a - b
a *= b	a = a*b
a /= b	a = a/b
a **= b	a = a**b
a %= b	a = a%b

Usando os Operadores

```
>>> s = 'Hello '
>>> t = 'to you'
>>> a = [1, 2, 3]
>>> print 3*s                  # Repetition
Hello Hello Hello
>>> print 3*a                  # Repetition
[1, 2, 3, 1, 2, 3, 1, 2, 3]
>>> print a + [4, 5]           # Append elements
[1, 2, 3, 4, 5]
>>> print s + t                # Concatenation
Hello to you
>>> print 3 + s                # This addition makes no sense
Traceback (most recent call last):
  File ''<pyshell#9>'', line 1, in ?
    print n + s
TypeError: unsupported operand types for +: 'int' and 'str'
```

Operadores

- Condicionais:

<	Menor que
>	Maior que
<=	Menor ou igual
>=	Maior ou igual
==	Igualdade (Comparação)
!=	Diferença (Comparação)

- Exemplos:

```
>>> a = 2              # Integer
>>> b = 1.99           # Floating point
>>> c = '2'            # String
>>> print a > b
1
>>> print a == c
0
>>> print (a > b) and (a != c)
1
>>> print (a > b) or (a == b)
1
```

Condicionais

- Estrutura **if**:
  - Construção: **if** *condição* **:**

    Bloco de Comandos

    - O "*Bloco de Comandos"* só será executado caso "*condição*" seja **verdadeira.** Caso contrário, será despresado.
    - A estrutura **if** pode ser aninhada e/ou composta por **elif** (else if) e/ou else.

```
if a < 0.0:
    sign = 'negative'
elif a > 0.0:
    sign = 'positive'
else:
    sign = 'zero'
return sign
```

Laços

- O laço **while**
  - Construção: **While** *condição* **:**

    Bloco de Comandos

```
nMax = 5
n = 1
a = []                  # Create empty list
while n < nMax:
    a.append(1.0/n)  # Append element to list
    n = n + 1
print a
```

The output of the program is

```
[1.0, 0.5, 0.3333333333333331, 0.25]
```

Laços

- O Laço **for**
  - Construção: **for** *alvo* **in** *sequencia*:

    Bloco de Comandos

```
nMax = 5
a = []
for n in range(1,nMax):
    a.append(1.0/n)
print a
```

```
list = ['Jack', 'Jill', 'Tim', 'Dave']
name = eval(raw_input('Type a name: '))  # Python input prompt
for i in range(len(list)):
    if list[i] == name:
        print name,'is number',i + 1,'on the list'
        break
else:
    print name,'is not on the list'
```

Lendo uma Entrada

- A função intrínseca para aceitar a entrada do usuário é:

```
raw_input(prompt)
```

  - Esta função abre um *prompt* na tela e ler o que o usuário digitar como uma *string*
- Para converter a *string* em um valor numérico, utiliza-se a função:

```
eval(string)
```

Exemplo

- Programa:

```
a = raw_input('Input a: ')
print a, type(a)            # Print a and its type
b = eval(a)
print b,type(b)             # Print b and its type
```

  - Saídas possíveis do programa:

```
Input a: 10.0
10.0 <type 'str'>
10.0 <type 'float'>

Input a: 11**2
11**2 <type 'str'>
121 <type 'int'>
```

Imprimindo um Objeto na Tela

- Para exibir um objeto na tela use a função:

```
print object1, object2, ...
```

  - Exemplo:

```
>>> a = 1234.56789
>>> b = [2, 4, 6, 8]
>>> print a,b
1234.56789 [2, 4, 6, 8]
>>> print 'a =',a, '\nb =',b
a = 1234.56789
b = [2, 4, 6, 8]
```

Controle de Exceções e Erros

- Quando um erro ocorre na execução de um programa uma exceção é levantada!
  - Estas exceções podem ser capturadas e tratadas:

```
try:
    do something
except error:
    do something else
```

Exemplo: Divisão por Zero

- Um erro possível em uma divisão numérica é a "divisão por zero":

```
>>> c = 12.0/0.0
Traceback (most recent call last):
  File ''<pyshell#0>'', line 1, in ?
    c = 12.0/0.0
ZeroDivisionError: float division
```

- Esta pode ser tratada como:

```
try:
    c = 12.0/0.0
except ZeroDivisionError:
    print 'Division by zero'
```

Funções

- Em Python é possível definir uma função com o comando **def**:

```
def func_name(param1, param2,...):
    statements
    return return_values
```

  - Se o comando **return** for omitido, então a função retornará um objeto nulo

Exemplo de Funções

- A função que calcula as primeira e segundas derivadas (aproximadamente):

```
def  finite_diff(f,x,h=0.0001):    # h has a default value
    df =(f(x+h) - f(x-h))/(2.0*h)
    ddf =(f(x+h) - 2.0*f(x) + f(x-h))/h**2
    return df,ddf
x = 0.5
df,ddf = finite_diff(arctan,x)    # Uses default value of h
print 'First derivative  =',df
print 'Second derivative =',ddf
```

- Saída do programa:

```
First derivative  = 0.799999999573
Second derivative = -0.639999991892
```

Módulos em Python

- Em Python existem vários **módulos** (bibliotecas) que contêm funções já implementadas
  - É possível invocar estes módulos como:
    - **from** *modulo* **import** *func1, func2, ...*
    - **from** *modulo* **import** *
    - **import** *modulo*
  - Com o comando **dir()** é possível verificar quais funções pertencem a um dado módulo:

```
>>> import math
>>> dir(math)
['__doc__', '__name__', 'acos', 'asin', 'atan',
 'atan2', 'ceil', 'cos', 'cosh', 'e', 'exp', 'fabs',
 'floor', 'fmod', 'frexp', 'hypot', 'ldexp', 'log',
 'log10', 'modf', 'pi', 'pow', 'sin', 'sinh', 'sqrt',
 'tan', 'tanh']
```

Alguns Exemplos Módulos

os	interface com o sistema operacional
shutil	manipulação de arquivos e diretórios.
glob	lista arquivos em diretório (caracteres curinga!)
email	manipulacao de e-mails
re	expressões regulares
math	funções matemáticas
pickle	salva dados da programa e recupera-os
imaging	processamento de imagens
urllib2	acesso a urls
smtplib	protocolo SMTP
gzip	compressão (entre outros: bz2, tarfile, zlib)
gtk	biblioteca gráfica
datetime	manipulação de datas e intervalos de tempo
timeit	teste de desempenho

Exercícios de Fixação

- Baixem e instalem o interpretador Python
- Pesquisem na internet GUI's para o Python
  - Observe que é possível compilar um programa sem o uso de uma janela gráfica:
    - Escreve-se o código em um editor de texto
    - Usa-se "python *programa.py*" para se compilar o código salvo no arquivo *programa.py*
- Escreva um programa em Python que ler dois números, determina se um é múltiplo do outro e imprime na tela esta informação com os dois números ordenados de forma crescente.