Inteligência Artificial

Aula 4

Profª Bianca Zadrozny

http://www.ic.uff.br/~bianca/ia-pos

Características de Python

- Gratuita. Roda em muitas plataformas.
  - Pode ser baixada em www.python.org
- Fácil de ler.
  - Ao contrário de Perl = "write only language"
- Tempo de implementação rápido.
  - Ao contrário de Java.
- Orientada a objeto.

Baixando e Instalando

- Baixar o Python 2.7 no site www.python.org
  - Para o Windows baixar o Python 2.7 Windows installer e instalar usando as opções default.
- Além do interpretador, será instalada uma IDE (IDLE).

Operadores

- O interpretador de Python pode ser usado para avaliar expressões aritméticas.

```
>>> 1 + 1
2
>>> 2 * 3
6
```

- Operadores booleanos podem ser usados para manipular os valores `True` e `False`.

```
>>> 1==0
False
>>> not (1==0)
True
>>> (2==2) and (2==3)
False
>>> (2==2) or (2==3)
True
```


Strings

- Como Java, Python tem um tipo string definido na própria linguagem. O operador + pode ser usado para concatenação.

```
 >>> 'inteligencia' + "artificial"
'inteligenciaartificial'
```

- Existem muitos métodos para manipular strings.

```
>>> 'artificial'.upper()
'ARTIFICIAL'
>>> 'HELP'.lower()
'help'
>>> len('Help')
4
```

- Podemos usar aspas simples ou duplas, facilitando o aninhamento de strings.

Variáveis

- Em Python não existe declaração de variáveis, simplesmente atribuímos valores a um nome e a variável com aquele nome passa a existir.

```
>>> s = 'hello world'
>>> print s
hello world
>>> s.upper()
'HELLO WORLD'
>>> len(s.upper())
11
>>> num = 8.0
>>> num += 2.5
>>> print num
10.5
```

Comandos de ajuda

- Para descobrir os métodos para um determinado tipo, podemos usar os comandos `dir` e `help`.

```
>>> s = 'abc'

>>> dir(s)
['__add__', '__class__', '__contains__', '__delattr__', '__doc__', '__eq__',
'__ge__', '__getattribute__', '__getitem__', '__getnewargs__', '__getslice__',
'__gt__', '__hash__', '__init__', '__le__', '__len__', '__lt__', '__mod__',
'__mul__', '__ne__', '__new__', '__reduce__', '__reduce_ex__', '__repr__',
'__rmod__', '__rmul__', '__setattr__', '__str__', 'capitalize', 'center',
'count', 'decode', 'encode', 'endswith', 'expandtabs', 'find', 'index',
'isalnum', 'isalpha', 'isdigit', 'islower', 'isspace', 'istitle', 'isupper',
'join', 'ljust', 'lower', 'lstrip', 'replace', 'rfind','rindex', 'rjust',
'rsplit', 'rstrip', 'split', 'splitlines', 'startswith', 'strip', 'swapcase',
'title', 'translate', 'upper', 'zfill']

>>> help(s.find)
Help on built-in function find:

find(...) S.find(sub [,start [,end]]) -> int Return the lowest index in S
where substring sub is found, such that sub is contained within s[start,end].
Optional arguments start and end are interpreted as in slice notation. Return
-1 on failure.
>> s.find('b')
1
```


Listas

- É uma estrutura de dados própria da linguagem para guardar sequências de itens.
- É **mutável**, isto é, podemos modificar elementos da lista individualmente.

```
>>> frutas = ['laranja','pera','banana']
>>> frutas[0]
   'laranja'
>>> frutas[1]= 'abacaxi‘
>>> frutas
['laranja', 'abacaxi','banana']
```


Listas (cont.)

- Podemos usar o operador + para concatenação de listas.

```
>>> outrasFrutas = ['kiwi', 'morango']
>>> frutas + outrasFrutas
>>> ['laranja', 'abacaxi', 'banana','kiwi', 'morango']
```

- Podemos usar índices negativos a partir do final da lista.

```
>>> frutas[-2]
‘abacaxi'
>>> frutas.pop()
'banana'
>>> frutas
['laranja', 'abacaxi']
>>> frutas.append('ameixa')
>>> frutas
  ['laranja', 'abacaxi', 'ameixa']
```


Indexação e Quebra

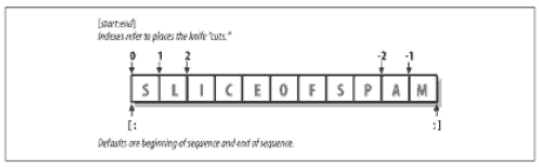

Figure 7-1. Offsets and slices: positive offsets start from the left end (offset 0 is the first item), and negatives count back from the right end (offset –1 is the last item). Either kind of offset can be used to give positions in indexing and slicing.

Listas (cont.)

- Podemos indexar pedaços de listas.

```
>>> frutas[0:2]
  ['laranja', 'abacaxi']
>>> frutas[:3]
  ['laranja', 'abacaxi', 'ameixa']
>>> frutas[1:]
  ['abacaxi', 'ameixa']]
>>> len(frutas)
  3
```


Listas (cont.)

- Listas podem conter elementos de qualquer tipo, inclusive outras listas.

```
>>> listaDeListas =
  [['a','b','c'],[1,2,3],['um','dois','tres']]
>>> listaDeListas[1][2]
  3
>>> listaDeListas[0].pop()
  'c'
>>> listaDeListas
  [['a', 'b'],[1, 2, 3],['um', 'dois', 'tres']]
```

Tuplas

- Tuplas são estruturas similares a listas, exceto que são imutáveis.
- Usa-se parênteses ao invés de colchetes.

```
>>> par = (3,5)
>>> par[0]
3
>>> x,y = par
>>> x
3
>>> y
5
>>> par[1] = 6
TypeError: object does not support item
  assignment
```


Conjuntos

- Conjuntos são estruturas de dados que armazenam listas não-ordenadas sem duplicatas.

```
>>> shapes = ['circle','square','triangle','circle']
>>> setOfShapes = set(shapes)
>>> setOfShapes
set(['circle','square','triangle'])
>>> setOfShapes.add('polygon')
>>> setOfShapes
set(['circle','square','triangle','polygon'])
>>> 'circle' in setOfShapes
True
>>> 'rhombus' in setOfShapes
False
>>> favoriteShapes = ['circle','triangle','hexagon']
>>> setOfFavoriteShapes = set(favoriteShapes)
>>> setOfShapes - setOfFavoriteShapes
set(['square','polyon'])
>>> setOfShapes & setOfFavoriteShapes
set(['circle','triangle'])
>>> setOfShapes | setOfFavoriteShapes
set(['circle','square','triangle','polygon','hexagon'])
```


Dicionários

- Dicionários são endereçados por *chave*, não por posição.
  - Podem ser vistos como uma coleção de pares chave:valor.
- São coleções *não-ordenadas* de objetos arbitrários.
- Tem tamanho *variável* e podem conter objetos de qualquer tipo, inclusive outros dicionários.
- São *mutáveis* como as listas.

Dicionários (cont.)

```
level = {'low':1, 'medium':5}
x = level['medium']           # 5
n = len(level)                # 2

flag = level.has_key('low')   # True
l = level.keys()              # ['low','medium']

level['low'] = 2      # {'low':2, 'medium':5}
level['high'] = 10    # {'low':2, 'high':10, 'medium':5}

level.items()
[('low',2), ('high',10), ('medium',5)]

level.values()
[2, 10, 5]
```


Execução Condicional

```
course = 'Syntax'
if course == 'Syntax':
  print 'Bhatt'
  print 'or Potts'
elif course == 'Computational Linguistics':
  print 'McCallum'
else:
  print 'Someone else'
```

- A indentação determina a estrutura do bloco.
  - É o único lugar onde o espaço em branco importa.
- A indentação ajuda na legibilidade do código.
- Expressões depois do if e elif podem ser de quase qualquer tipo.
  - False, 0, [], (), '' funcionam como falso, o resto é verdadeiro.

Laços "While"

- Um laço do tipo "while" continua enquanto a expressão no topo for verdadeira.

```
a = 0
b = 10
while a < b:
  print a
  a = a + 1
```

```
s = 'abcdefg'
while len(s) > 0:
  print s
  s = s[1:]
```

Laços "For"

- "For" é usado pra percorrer uma seqüência qualquer de objetos.

```
l = ['a', 'b', 'c']
for i in l:
  print i

sum = 0
for x in [1, 2, 3, 4, 5, 6]:
  sum = sum + x
print sum
```

- O uso de "range" pode ser útil.

```
range(5)        # [0, 1, 2, 3, 4]
range(2,5)      # [2, 3, 4]
range(0,6,2)    # [0, 2, 4]
```


Laços "For"

- Fazer alguma coisa com cada item de uma lista.

```
l = [1, 2, 3, 4, 5, 6]  # or l = range(1,7)

# one way to print the square
for x in l:
  print x*x

# another way to do it
n = len(l)
for i in range(n):
  print l[i]*l[i]
```


Exemplo: Interseção

```
l1 = ['a', 'd', 'f', 'g']
l2 = ['a', 'b', 'c', 'd']
# one way
result = []
for x in l1:
  for y in l2:
      if x == y:
          result.append(x)
# or, alternatively
result = []
for x in l1:
  if x in l2:
      result.append(x)        # result == ['a', 'd']
```


Funções "built-in", importadas e definidas pelo usuário

- "Built-in"

```
l = len(['a', 'b', 'c'])
```

- Importadas

```
import math
from os import getcwd
print getcwd() # which directory am I in?
x = math.sqrt(9)    # 3
```

- Definidas pelo usuário

```
def multiply(a, b):
     return a * b
print multiply(4,5)
print multiply('-',5)
```


Definição de funções

- **`Def`** cria um objeto do tipo função e dá um nome a ele.
- **`Return`** retorna um objeto a quem chamou a função.

```
def square(x):       # create and assign
     return x*x
y = square(5)        # y gets 25
```


Exemplo

```
def intersect(seq1, seq2)
     result = []
     for x in seq1:
          if x in seq2:
               result.append(x)
     return result
```

Variáveis locais

- Variáveis dentro de uma função são locais àquela função.

```
>>> intersect(s1, s2):
...   result = []
...   for x in s1:
...         if x in s2:
...               result.append(x)
...   return result
...
>>> intersect([1,2,3,4], [1,5,6,4])
[1, 4]
>>> result
Traceback (most recent call last):
  File "<stdin>", line 1, in ?
NameError: name 'result' is not defined
```


Passagem de Parâmetros

- Objetos imutáveis são passados "por valor".

```
>>> def plusone(x):
...   x = x + 1
...   return x
...
>>> plusone(3)
4
>>> x = 6
>>> plusone(x)
7
>>> x
6
```


Passando parâmetros mutáveis

- Números, strings e tuplas são imutáveis enquanto listas e dicionários são mutáveis.
- Objetos mutáveis são passados "por referência".

```
>>> def appendone(s):
...   s.append('one')
...   return s
...
>>> appendone(['a', 'b'])
['a', 'b', 'one']
>>> l = ['x', 'y']
>>> appendone(l)
['x, 'y', 'one']
>>> l
['x', 'y', 'one']
```


Número variável de parâmetros

```
def max (*a):
  maximum = 9999999
  for x in a:
      if x > maximum:
          maximum = x
  return maximum
```


Parâmetros opcionais

```
def exp (x, exponent=2.718):
  return exponent ** x
```

```
>>> exp(1)
2.718
>>> exp(1, 2.0)
2.0
>>> exp(3, 2.0)
8.0
>>> exp(3, exponent=2.0)
8.0
```


Múltiplos parâmetros opcionais

```
def exp_plus (x, exponent=2.718, addend=0):
  return (exponent ** x) + addend
```

```
>>> exp(1)
2.718
>>> exp(1, 2.0)
2.0
>>> exp(1, exponent=2.0)
2.0
>>> exp(1, addend=2.0)
4.718
```

Número arbitrário de parâmetros opcionais

- A notação ** recebe todos os parâmetros extra em um dicionário.

```
def showargs (separator, **d):
  for key in d.keys():
      print str(key)+":"+str(d[key])+separator,
  print
```

```
>>> showargs(";", bi=2, tri=3, quad=4)
tri:3;bi:2;quad:4;
```

```
def showargs (separator, **d):
  for (key,val) in d.items():
      print str(key)+":"+str(val)+separator,
  print
```


Definição de Classes

```
class FruitShop:

    def __init__(self, name, fruitPrices):
        """
            name: Name of the fruit shop

            fruitPrices: Dictionary with keys as fruit
            strings and prices for values e.g.
            {'apples':2.00, 'oranges': 1.50, 'pears': 1.75}
        """
        self.fruitPrices = fruitPrices
        self.name = name
        print 'Welcome to the %s fruit shop' % (name)

    def getCostPerPound(self, fruit):
        """
            fruit: Fruit string
        Returns cost of 'fruit', assuming 'fruit'
        is in our inventory or None otherwise
        """
        if fruit not in self.fruitPrices:
            print "Sorry we don't have %s" % (fruit)
            return None
        return self.fruitPrices[fruit]

    def getPriceOfOrder(self, orderList):
        """
            orderList: List of (fruit, numPounds) tuples

        Returns cost of orderList. If any of the fruit are
        """
        totalCost = 0.0
        for fruit, numPounds in orderList:
            costPerPound = self.getCostPerPound(fruit)
            if costPerPound != None:
                totalCost += numPounds * costPerPound
        return totalCost

    def getName(self):
        return self.name
```


Usando objetos

- Supõe-se que a definição das classes está no arquivo `shop.py`.

```
import shop

shopName = 'the Berkeley Bowl'
fruitPrices = {'apples': 1.00, 'oranges': 1.50, 'pears': 1.75}
berkeleyShop = shop.FruitShop(shopName, fruitPrices)
applePrice = berkeleyShop.getCostPerPound('apples')
print applePrice
print('Apples cost $%.2f at %s.' % (applePrice, shopName))

otherName = 'the Stanford Mall'
otherFruitPrices = {'kiwis':6.00, 'apples': 4.50, 'peaches': 8.75}
otherFruitShop = shop.FruitShop(otherName, otherFruitPrices)
otherPrice = otherFruitShop.getCostPerPound('apples')
print otherPrice
print('Apples cost $%.2f at %s.' % (otherPrice, otherName))
print("My, that's expensive!")
```


Variáveis Estáticas

person_class.py

```
class Person:
    population = 0
    def __init__(self, myAge):
        self.age = myAge
        Person.population += 1
    def get_population(self):
        return Person.population
    def get_age(self):
        return self.age
```

```
>>> import person_class
>>> p1 = person_class.Person(12)
>>> p1.get_population()
1
>>> p2 = person_class.Person(63)
>>> p1.get_population()
2
>>> p2.get_population()
2
>>> p1.get_age()
12
>>> p2.get_age()
63
```