Universidade de Aveiro

Introdução à Inteligência Artificial – MIECT
Inteligência Artificial – LEI

Programação ao Estilo Funcional em Python

Ano lectivo 2015/2016

Regente: Luís Seabra Lopes

Python

- Características principais da linguagem de programação Python:
  - Interpretada
  - Interactiva
  - Portável
  - Funcional
  - Orientada a objectos
  - Implementação aberta

Python (cont.)

- Objectivos da linguagem
  - Simplicitade sem prejuizo da utilidade
  - Programação modular
  - Legibilidade
  - Desenvolvimento rápido
  - Facilidade de integração, nomeadamente com outras linguagens

Python é multi-paradigma

Programação funcional

Expressões lambda
Funções de ordem superior
Listas com sintaxe simplificada
Listas de compreensão
Iteradores

Programação OO

Classes
Objectos
Métodos
Herança

Programação imperativa / modular

Instrução de atribuição
Sequências de instruções
Análise condicional (if-elif-else)
Ciclos for, while
Sistema de módulos

Python - história

- Criada em 1989-1991 por Guido Van Rossum
  - Tem como predecessora directa a linguagem imperativa/estruturada ABC, tendo também sido influenciada pela linguagem Modula-3
  - O nome da linguagem tem origem no "*Monty Python's Flying Circus*"
- Inicialmente desenhada como uma linguagem de *scripting* no sistema operativo Amoeba

Python versus Java

- Código mais conciso
  - Os espaços brancos são sintaticamente relevantes
- Verificação de tipos dinâmica
- Desenvolvimento mais rápido
- Não compila para código nativo

- Mas, programas mais lentos …

Python – áreas de aplicação

- Interligação de sistemas
- Aplicações gráficas
- Aplicações para bases de dados
- Multimédia
- Internet protocol / Web
- Robótica & inteligência artificial

Python - aplicações

- Google - fortemente baseado em Python, segundo o lema inicial:
  - "Python where we can, C++ where we must"
- Zope – servidor de aplicações para a Web, de código aberto, totalmente escrito em Python
- ROS (Robot Operating System) – Python é uma das linguagens suportadas, juntamente com C++ e Lisp

Python – aplicações (cont.)

- Inteligência Artificial – Python é uma das linguagens com popularidade crescente nesta área
  - Os exemplos do livro "*Artificial Intelligence: a Modern Approach*" estão implementados em Python, Java e Lisp
  - CWM – Máquina de inferência de propósito geral para a Web Semântica, totalmente desenvolvido em Python por Tim Berners-Lee
  - Google (algoritmo de ordenação de páginas, etc.) também é um exemplo aqui!

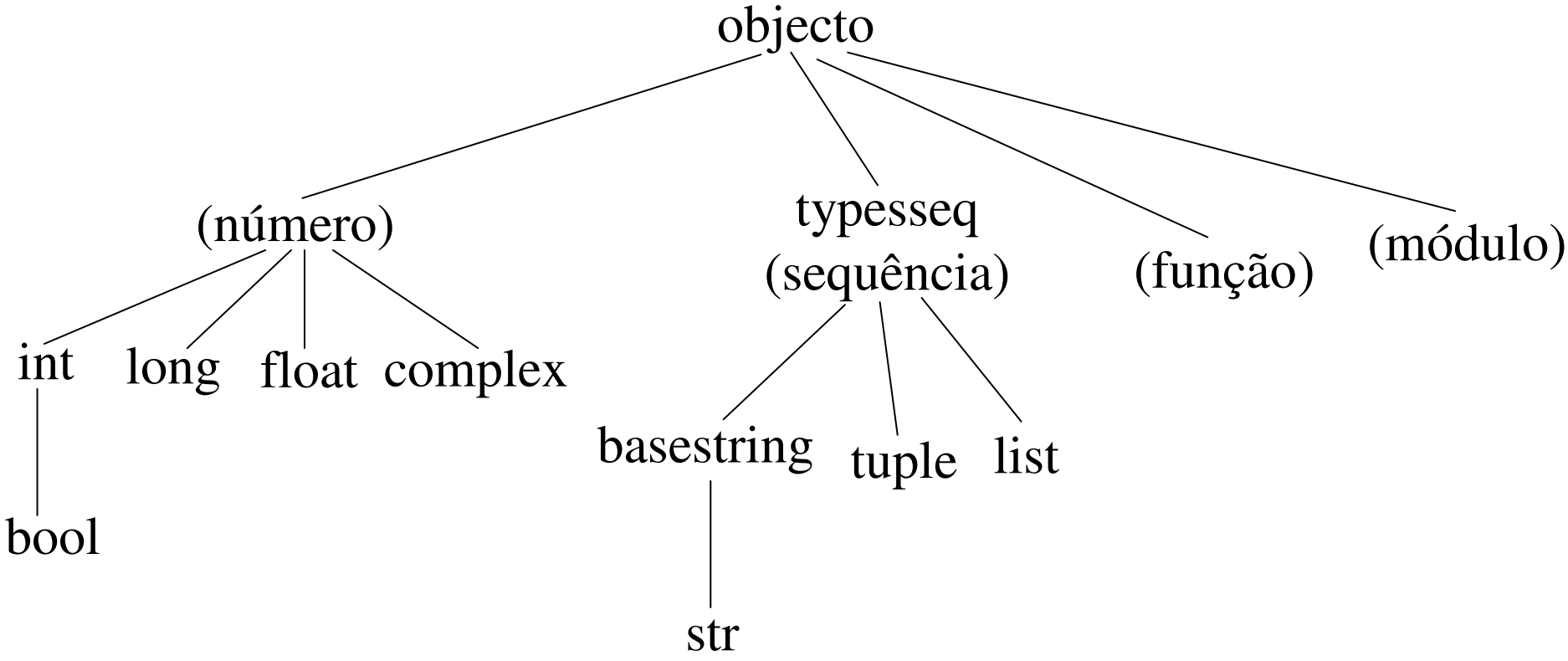
Pyhton - utilização

- Está incluída nas principais distribuições de Linux
  - Está também disponível para outras plataformas
- Pode ser obtida, juntamente com documentação em www.python.org
- IDLE – ambiente de desenvolvimento para Python

Dados, ou “objectos”

- Objecto – no contexto de Pyhton, esta designação é aplicada a qualquer dado que possa ser armazenado numa variável, ou passado como parâmetro a uma função
- Cada objecto é caracterizado por
  - Identidade ou referência (identifica a posição de memória onde está armazenado),
  - Tipo e
  - Valor
- Alguns tipos de objectos podem ter atributos e métodos
- Alguns tipos (classes) de objectos podem ter sub-tipos (sub-classes)

Objectos


objecto
(número)
typesseq
(sequência)
(função)
(módulo)
int
long
float
complex
bool
basestring
tuple
list
str

Tipos de dados elementares

- bool
  - Tem os valores True e False
  - Mas, valores de diferentes tipos podem também ser usados como valores de verdade
    - Exemplo: o inteiro 0 (zero) vale o mesmo que False
  - Função bool() converte qualquer valor para bool

Tipos de dados elementares (cont.)

- Números
  - int – números inteiros de 32 bits
    - equivale ao long da linguagem C
  - long – inteiros de precisão ilimitada
  - float – números reais
    - equivale ao double da linguagem C
  - complex – números reais, em que a parte real e a parte imaginária são números reais
    - Exemplos: 1.5+0.5j, 7.2+4.0J
    - As partes de um número complexo z podem ser obtidas através das expressões z.real e z.imag

Tipos de dados elementares (cont.)

- Números (cont.)
  - Funcionam com os operadores habituais: +, -, *, /
  - Quociente da divisão inteira: //
  - O operador de divisão (/) também fornece um quociente da divisão inteira, mas neste caso o resultado é arredondado para o lado de $-\infty$
    - Exemplos: 1/2 → 0, 1/(-2) → -1
  - Resto da divisão inteira: %
  - Potência: ** (equivale à função pré-definida pow())
    - Exemplo: 5 ** 3 → 125
  - Funções de conversão:
    - int(), long(), float()

Sequências de dados

- Cadeias de caracteres (str)
  - Podem aparecer entre aspas (") ou pelicas (')
  - Exemplos:
    - "abc d x"
    - 'abc d x'
    - 'Ele disse "sim" !'
    - "A palavra 'Maria' é um nome próprio."
  - As cadeias de caracteres são imutáveis: não podemos modificar os caracteres em posições individuais

Sequências de dados (cont.)

- Tuplos (tuple) – agregados ou composições de vários elementos, que podem ser de tipos diferentes
  - Funcionam como registos ou estruturas sem nome
  - São imutáveis: não podemos modificar elementos em posições individuais do tuplo
  - Os elementos são separados por vírgulas (,) e opcionalmente delimitados por parênteses curvos
  - Exemplos:
    - 1, 2, 'a'
    - ("maria", 33)
    - 27,
    - 'lisboa', ("colinas", 7)
    - ( )

Sequências de dados (cont.)

- Listas (list) – sequências de elementos, que podem ser de tipos diferentes
  - Combinam a funcionalidade usual das listas na programação declarativa com a funcionalidade usual dos vectores na programação imperativa
  - É possível modificar elementos individuais das listas
  - Os elementos são separados por vírgulas (,) e delimitados por parênteses rectos
  - Exemplos:
    - [1, 2, 'a']
    - [("maria", 33), (“josé", 40)]
    - ['lisboa', [7, "colinas"]]
    - [ ]

Sequências de dados (cont.)

- Algumas funcionalidades básicas
  - `x in s`
    - Expressão que retorna `True` se existir um elemento na sequência `s` que seja igual a `x`, caso contrário retorna `False`
  - `x not in s`
    - O contrário da anterior
  - `len(s)`
    - Função que retorna o comprimento da sequência `s`
  - `s1 + s2`
    - Retorna a concatenação das sequências `s1` e `s2`

Função pré-definida `type()`

- Dado um objecto qualquer, devolve o respectivo tipo

Variáveis

- Não são declaradas
- Não têm tipo
- Praticamente tudo pode ser atribuído a uma variável (incluindo funções, módulos e classes)
- Similarmente ao que acontece nas linguagens imperativas, e ao contrário do que acontece nas linguagens funcionais, em Python o valor das variáveis pode ser alterado
- Não se pode ler o valor da variável se ela não tiver sido inicializada

Instrução de atribuição - I

- Tal como é habitual na programação imperativa, e ao contrário do habitual na programação declarativa, Python possui instrução de atribuição
- Exemplos:
  - `n = 10`
  - `a = b = c = 0`
  - `x = 7.25`
  - `cad = "cadeia"`
  - `t = (n, x)`
  - `lista = [1, 2, 'quatro', 8.0]`

Instrução de atribuição - II

- Podemos usar a operação de atribuição para decompor estruturas
- Exemplo:
  - `triplo = (1, 2, 3)`
  - `(i, j, k) = triplo`
    - Como resultado, `i=1`, `j=2`, `k=3`
- Outro exemplo:
  - `(q,r) = divmod(16,3)`
    - A função pré-definida `divmod()` devolve um tuplo com o quociente e o resto de uma divisão inteira
    - Como resultado, `q=5`, `r=1`

Operadores de comparação

- Igual: ==
- Diferente: != (ou <>)
- Menor/maior: <, <=, >, >=
- Objectos de tipos diferentes nunca são iguais
  - Exceptuam-se os diferentes tipos de números
- Comparação de sequências baseia-se num critério lexicográfico
- Mais info: *Python Tutorial*, sec. 5.8

Operadores lógicos

- Conjunção: `and`
- Disjunção: `or`
- Negação: `not`

- Nota: Na conjunção e na disjunção, o segundo argumento só é avaliado se for necessário para determinar o resultado

Acesso a sequências

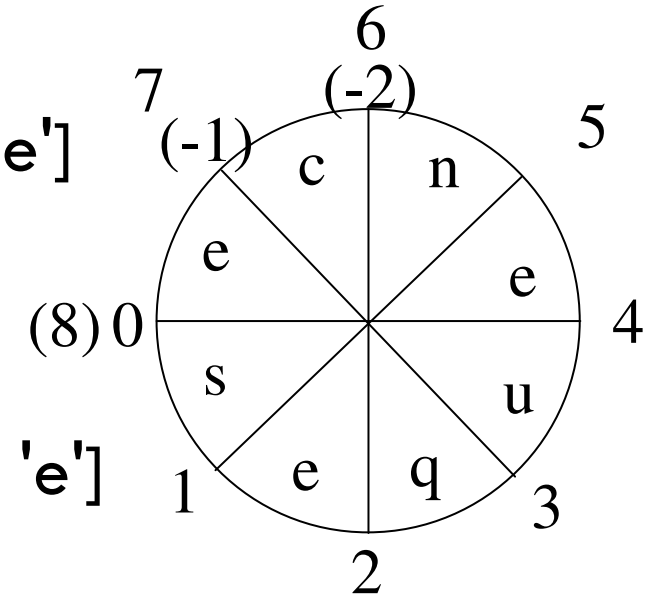
- Os elementos das sequências são acedidos através de índices inteiros consecutivos
  - O primeiro elemento tem o índice 0 (zero)
- Exemplo:
  - a = [12, 4, "abc"]
  - a[0] → 12
  - a[2] → "abc"

Acesso a sequências (cont.)

- É possível extrair "fatias" das sequências
  - Formato: `seq[inf:sup]` – fatia da sequência `seq`, compreendida entre o elemento com índice `inf` e o elemento com índice `sup-1`
  - A fatia é uma cópia do conteúdo da sequência original entre `inf` e `sup-1`
- A indexação é circular, o que permite aceder ao último elemento da sequência `s` pelo índice `len(s)-1` ou simplesmente pelo índice `-1`

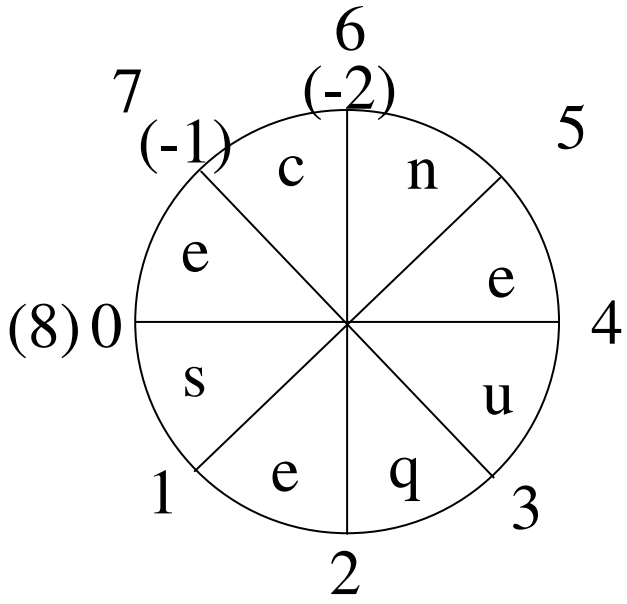
Acesso a sequências (cont.)

- Exemplos:
  - lista = ['s', 'e', 'q', 'u', 'e', 'n', 'c', 'e']
  - lista[0] → 's'
  - lista[2:5] → ['q', 'u', 'e']
  - lista[1:] → ['e', 'q', 'u', 'e', 'n', 'c', 'e']
  - lista[-2:] → ['c', 'e']
  - lista[:3] → ['s', 'e', 'q']
  - lista[:] → ['s', 'e', 'q', 'u', 'e', 'n', 'c', 'e']



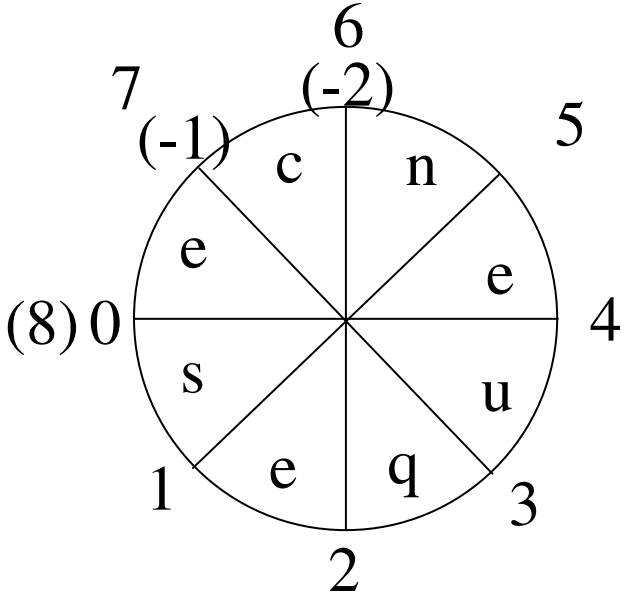
Modificação de listas

- Faz-se por atribuição
  - lista[0] = 'p'
  - lista → ['p', 'e', 'q', 'u', 'e', 'n', 'c', 'e']
- Pode-se modificar fatias
  - lista[-2:] = ['o', 's']
  - lista → ['p', 'e', 'q', 'u', 'e', 'n', 'o', 's']



Modificação de listas (cont.)

- Pode-se remover uma fatia
  - lista[0:3] = [ ]
  - lista → ['u', 'e', 'n', 'o', 's']
- Inserir uma fatia
  - lista[3:3] = ['a', 'a']
  - lista → ['u', 'e', 'n', 'a', 'a', 'o', 's']
- Substituir por fatia de tamanho diferente
  - lista[1:6] = ['c', 'h']
  - lista → ['u', 'c', 'h', 's']



Instrução de atribuição - III

- Detalhe importante:
  - A instrução de atribuição, em vez de copiar valores, limita-se a associar um dado identificador a um dado objecto
  - Assim, a atribuição de uma variável `x` a uma variável `y` apenas tem como resultado associar `y` ao mesmo objecto ao qual `x` já estava associada
  - No caso de objectos mutáveis, há que ter cuidado com efeitos como este:
    - `a = [1,2,3]`
    - `b = a`
    - `b[1:2] = []`
    - `a` → `[1,3]`

Análise condicional: instrução if-elif-else

- Síntaxe:

```
if <condição_1>:
        <instruções_1>
elif <condição_2>:
        <instruções_2>
else:
        <instruções_n>
```

- Notas:
  - Pode haver 0 (zero) ou mais ramos elif (= else if)
  - O ramo elif / else é opcional

Análise condicional: expressão if-else

- Síntaxe
  - <*expressão1*> if <*condição*> else <*expressão2*>
- Exemplos:
  - x if x>=0 else -x
  - 'pos' if a>0 else 'nul' if a==0 else 'neg'

Definição de funções

- Inicia-se com a palavra `def`
- Passagem de parâmetros "por referência dos objectos" (ver adiante)
- A instrução `return` permite definir o resultado da função
  - Quando não é indicado um resultado no `return`, ou quando não há `return`, o resultado é `None` (um identificador pré-definido)
  - Funções sem `return`, ou com `return` vazio, são também conhecidas como "*procedimentos*"
- As funções podem ser recursivas
- As funções são objectos do tipo `function`

Definição de funções – exemplos (I)

```
# retorna o primeiro elemento de uma lista, caso exista,
# ou None, caso contrário
def cabeca(lista):
    if lista==[]:
        return None
    return lista[0]

# retorna um tuplo com o primeiro elemento de uma lista
# e a lista dos restantes (retorna None caso a lista seja vazia)
def cabeca_e_cauda(lista):
    if lista==[]:
        return None
    return (lista[0],lista[1:])
```

Definição de funções – exemplos (II)

```
# concatena duas listas
def concatenar(lista1,lista2):
    conc = lista2[:]
    conc[:0] = lista1
    return conc
```

```
# Nota: a concatenação de listas é disponibilizada
# em Python através do operador +
```

Passagem de parâmetros

- Os parâmetros são passados às funções segundo um mecanismo de "passagem por valor" ("*call by value*")
- Mas, detalhe importante: Neste contexto, o valor é na verdade a referência do objecto!!!
  - Se atribuirmos um novo objecto a uma variável passada por parâmetro, essa atribuição ocorre apenas no espaço de nomes da função (acetato seguinte, esquerda)
  - Se modificarmos um objecto passado por parâmetro (por exemplo, apagar um elemento de uma lista), isto não altera a referência do objecto, e portanto vai permanecer após o retorno da função (acetato seguinte, direita)

Passagem de parâmetros – exemplos (I)

```
>>> def incr(x):
...     x=x+1
...     return x
...
>>> n=10
>>> incr(n)
11
>>> n
10
```

```
>>> def acresc(l,x):
...     l[0:0]=[x]
...     return l
...
>>> lista=[5,12]
>>> acresc(lista,30)
[30,5,12]
>>> lista
[30,5,12]
```

Passagem de parâmetros – exemplos (II)

- O problema anterior resolve-se trabalhando sobre uma cópia local
- No caso de uma lista l, a fatia l[:] dá-nos uma cópia integral

```
>>> def acresc(l,x):
...     aux=l[:]
...     aux[0:0]=[x]
...     return aux
...
>>> lista=[5,12]
>>> acresc(lista,30)
[30,5,12]
>>> lista
[5,12]
```

Parâmetros com valores por defeito

- Exemplo:

```
def custoCombustivel(dist,cons=8,preco=1.5):
    return (dist/100.0) * cons * preco
```

- Na chamada, pode-se omitir alguns parâmetros, caso em que temos que indicar os nomes dos que estão para a frente:

```
>>> custoCombustivel(30,preco=1.6)
2.56
```

- Nas chamadas em que são fornecidos os primeiros *k* de *n* parâmetros, não é preciso indicar os nomes:

```
>>> custoCombustivel(20,7.5)
2.25
```

Funções recursivas – exemplos - I

```
# devolve factorial de um número n
def factorial(n):
    if n==0:
        return 1
    if n>0:
        return n*factorial(n-1)

# devolve o comprimento de uma lista
def comprimento(lista):
    if lista==[]:
        return 0
    return 1+comprimento(lista[1:])
```

```
    comprimento([1,2,3])
=
    comprimento([1|[2,3]])
=
    1 + comprimento([2,3])
=
    1 +( 1 + comprimento([3]))
=
    1 + (1 + (1 + comprimento([])))
=
    1 + (1 + (1 + 0))
=
    1 + (1 + 1 )
=
    1 + 2
=
    3
```

Funções recursivas – exemplos - II

```
# verifica se um elemento é membro de uma lista
def membro(x,lista):
    if l==[]:
        return False
    return (lista[0]==x) or membro(x,lista[1:])

# devolve uma lista com os elementos da lista
# de entrada por ordem inversa
def inverter(lista):
    if lista==[]:
        return []
    inv = inverter(lista[1:])
    inv[len(inv):] = [lista[0]]
    return inv
```

Expressões Lambda – I

- São expressões cujo valor é uma função
- São um "ingrediente" clássico da programação funcional
- Exemplos:
  - `lambda x : x+1`
    - Função que dado um valor `x`, devolve `x+1`
  - `m = lambda x, y : math.sqrt(x**2+y**2)`
    - Função que calcula o módulo de um vector (x,y), função esta atribuída à variável `m`
  - `(lambda lista : lista[-1]-lista[0]) [5,7,11,19,38]`
    - Função que calcula a diferença entre o primeiro e o último elemento de uma lista, função esta logo aplicada a uma lista concreta
    - Resultado: `33`

Expressões Lambda – II

- Como qualquer objecto, uma expressão lambda pode ser passada como parâmetro a uma função
- Exemplo:
  - Uma função h que, dada uma função f e um valor x, produz f(x)*x

```
def h(f,x): return f(x)*x
```

  - Exemplo de utilização:

```
h(lambda x : x+1,7)
```

  - Resultado: 56

Expressões Lambda – III

- As expressões lambda podem ser produzidas por outras funções:
  - Exemplo: Dado um inteiro n, a função seguinte produz uma função que soma n à sua entrada

```
def faz_incrementador(n):
    return lambda x : x+n
```

  - Exemplo de utilização:

```
suc = faz_incrementador(1)
suc(10)
```

  - Resultado: 11

Expressões Lambda – IV

- As expressões lambda são também conhecidas como expressões funcionais
- As funções que recebem expressões lambda como entrada e/ou produzem expressões lambda como saída são conhecidas como funções de ordem superior

- Nota importante: As expressões lambda só são úteis enquanto são simples. Uma função complexa merece ser escrita de forma clara numa definição (`def`) à parte

Exercício

- Pesquisa dicotómica de uma raiz de uma função *f* num intervalo [*a*,*b*]
  - Assume-se que a função é contínua em [*a*,*b*]
  - Assume-se que *f*(*a*) e *f*(*b*) são de sinais opostos
  - Implementa-se uma função que divide ao meio o intervalo e se chama a si própria recursivamente sobre a metade do intervalo em cujos extremos *f* tem valores de sinal contrário
  - A função *f* é um parâmetro de entrada da função que procura a raiz
  - O processo termina quando o valor *b*-*a* for suficientemente pequeno

Aplicar uma função a uma lista

- Aplicar uma função f a cada um dos elementos de uma lista, devolvendo uma lista com os resultados:

```
def aplicar(f,lista):
    if lista=[]:
        return []
    return [f(lista[0])] + aplicar(f,lista[1:])
```

- Exemplo de utilização: Dada uma lista de inteiros, obter a lista dos dobros

```
aplicar(lambda x : 2*x, [2,-4,17])
```

  - Resultado: [4,-8,34]

- Corresponde à função pré-definida map()

Filtrar uma lista

- Dada uma função booleana f e uma lista, devolve uma lista com os elementos da lista de entrada para os quais f devolve True:

```
def filtrar(f,lista):
    if lista=[]:
        return []
    if f(lista[0]):
        return [lista[0]] + filtrar(f,lista[1:])
    return filtrar(f,lista[1:])
```

- Exemplo: Dada uma lista de inteiros, obter a lista dos pares

```
filtrar(lambda x : x%2==0, [2,-4,17])
```

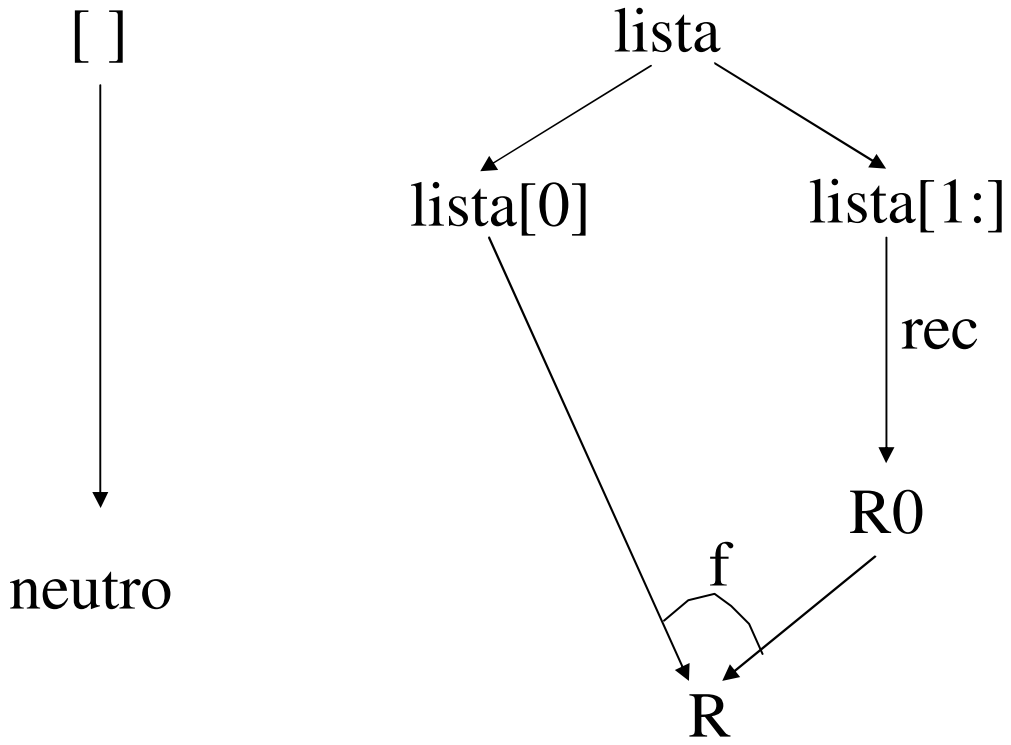
  - Resultado: [2,-4]

- Corresponde à função pré-definida filter()

Reduzir uma lista a um valor - I

- Muitos procedimentos que actuam sobre listas têm em comum a seguinte estrutura:
  - No caso de a `lista` ser vazia, o resultado é um valor "neutro" pré-definido;
  - No caso de a `lista` ser não vazia, o resultado da função depende de combinar a cabeça da lista (`lista[0]`) com o resultado da chamada recursiva sobre os restantes elementos (`lista[1:]`).

Reduzir uma lista a um valor - II



Exemplo:
Se quisermos somar os elementos de uma lista de inteiros:

```
neutro = 0
f = lambda x, s : x+s
```

Reduzir uma lista a um valor - III

- Dada uma função de combinação f, uma lista e um valor neutro, devolve a redução da lista:

```
def reduzir(f,lista,neutro):
    if lista=[]:
        return neutro
    return f(lista[0],reduzir(f,lista[1:],neutro))
```

- Exemplo: Dada uma lista de inteiros, obter a respectiva soma

```
reduzir(lambda x, s : x+s, [2,-4,17], 0)
```

  - Resultado: 15

- Corresponde à função pré-definida reduce()

Listas de compreensão

- Do inglês "*list comprehension*"
- Mecanismo compacto para processar alguns ou todos os elementos numa lista
  - "importado" da linguagem funcional Haskell
  - Pode ser aplicado a listas, tuplos e cadeias de caracteres
  - O resultado é uma lista
- Síntaxe (caso simples):

  [<expr> for <var> in <sequência> if <condição>]

Listas de compreensão (cont.)

- Podem funcionar como a função map()
- Exemplo: Obter os quadrados dos elementos de uma dada lista:

```
>>> map(lambda x : x**2, [2,3,7])
[4,9,49]
>>> [x**2 for x in [2,3,7]]
[4,9,49]
```

Listas de compreensão (cont.)

- Podem funcionar como a função filter()
- Exemplo: Obter os elementos pares existentes numa dada lista

```
>>> filter(lambda x : x%2==0, [2,3,7,6])
[2,6]
>>> [x for x in [2,3,7,6] if x%2==0]
[2,6]
```

Listas de compreensão (cont.)

- Podem combinar as funcionalidades de map() e filter()
- Exemplo: Obter os quadrados de todos os elementos positivos de uma dada lista

```
>>> [x**2 for x in [3,-7,6] if x>0]
[9,36]
```

Listas de compreensão (cont.)

- Podem percorrer várias sequências
- Exemplo: Obter todos os pares de elementos, um de uma lista e outro de outra, em que a soma seja ímpar

```
>>> [(x,y) for x in [3,7,6]
...             for y in [2,8,9] if (x+y)%2!=0]
[(3,2), (3,8), (7,2), (7,8), (6,9)]
```

Classes

- As classes em Python possuem as características mais comuns nas linguagens orientadas a objectos
  - Uma classe define um conjunto de objectos caracterizados por diversos atributos e métodos
  - É possível definir hierarquias de classes com herança
- As classes surgem na linguagem Python com pouca sintaxe adicional

Classes (cont.)

- Sintaxe:

```
class <nome-classe>:
    <declaração-1>
    ...
    <declaração-N>
```

Classes – exemplo

```
class UmTeste:
    def dizer_ola(self):
        print "Ola"
```

Por convenção, as palavras no nome de uma classe iniciam-se com maiúscula

Exemplo de definição de um método

- Utilização

```
>>> x = UmTeste()
>>> x.dizer_ola()
Ola
```

Criação de uma instância e atribuição a uma variável

Invocação do método

Classes com construtor – exemplo

```
class Complexo:
    def __init__(self,real,imag):
        self.r = real
        self.i = imag
```

O construtor é o método que inicializa um objecto no momento da sua criação; chama-se obrigatoriamente "__init__";

O primeiro parâmetro (self) de qualquer método é a própria instância na qual o método é chamado

- Utilização

```
>>> c = Complexo(-1.5,13.1)
>>> c.r, c.i
(-1.5,13.1)
```

Criação de uma instância e atribuição a uma variável

Classes – atributos

- No exemplo anterior, a classe `Complexo` tem os atributos `r` e `i`
- Tal como acontece com as variáveis normais, também os atributos das classes não são declarados
- Acesso aos atributos numa instância é feito com o ponto (“.”), como no exemplo anterior
- A todo o tempo, pode-se criar um novo atributo numa instância, bastando para isso atribuir-lhe um valor

Classes derivadas / herança

- Sintaxe:

  class <*nome-classe*> (<nome-classe-mãe>):

  <*declaração-1*>

  …

  <*declaração-N*>

- A classe derivada herda os métodos e atributos da classe mãe
- É possível uma classe ter várias classes mães

Exemplo de aplicação: expressões aritméticas

- Considere a seguinte expressão:

  2*x+1

- Pode-se representar em Python da seguinte forma:

  Soma(Produto(Const(2),Var()),Const(1)

- Em que Soma, Produto, Const e Var são classes definidas pelo programador para representar
  - somas de expressões,
  - produtos de expressões,
  - constantes e
  - ocorrências da variável

Exemplo de aplicação: expressões aritméticas (cont.)

- Como definir os construtores das classes referidas?
- Como definir métodos para avaliar as expressões, dado um certo valor da variável?
- Como definir métodos para simplificar expressões?
- Como definir métodos para derivar expressões?

Exemplo de aplicação: expressões aritméticas (cont.)

- Exemplo:

```
class Soma:
    def __init__(self,e1,e2):
        self.arg1 = e1
        self.arg2 = e2
    def avaliar(self,v):
        return self.arg1.avaliar(v) + self.arg2.avaliar(v)
```

Classes – conversão para cadeia de caracteres

- Relevante para visualização
- Consegue-se através da implementação de um método "__str__()" (nome obrigatório)
- Na classe Soma (ver acetato anterior), poderia ser assim:

```
def __str__(self):
    return str(self.arg1) + "+" + str(self.arg2)
```

- Utilização:

```
>>> s = Soma(Const(2),Const(1))
>>> str(s)
2+1
```

Métodos e atributos pré-definidos

- Métodos
  - __init__() – construtor
  - __str__() – define a uma conversão "informal" para cadeia de caractéres; suporta a função de conversão str()
  - __repr__() – define a representação "oficial" em cadeia de caractéres; suporta a função repr()
- Atributos
  - __class__ - identifica a classe de um dado objecto
    - Também se pode usar a função isinstance(<instance>,<class>)

O tipo list de Python é uma classe

- Tem os seguintes métodos:
  - list.append(x) – acrescenta x ao fim da lista
  - list.extend(L) – acrescenta elementos da lista L no fim da lista
  - list.insert(i,x) – insere x na posição i
  - list.remove(x) – remove a primeira ocorrência de x
  - list.index(x) – remove a posição da primeira ocorrência de x
  - list.sort() – ordena a lista (modifica a lista)
  - …