IA – POWERPOINT 3 – RESUMO

**Principais características da linguagem de programação Python:**

- Interpretada

- Interativa

- Portável

- Funcional

- Orientada a Objetos

- Implementação aberta

**Objetivos da linguagem:**

- Simplicidade sem prejuízo 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

- Objetos

- Métodos

- Herança

- Programação imperativa / modular

- Instrução de atribuição

- Sequências de instruções

- Análise condicional (if-elif-else)

- Ciclos for, while

- Sistema de módulos

**Python vs Java**

- Código mais conciso

- Verificação de tipos dinâmica

- Desenvolvimento mais rápido

- Não compila para código nativo

- Porém, códigos mais lentos

**Python – áreas de aplicação**

- Interligação de sistemas

- Aplicações gráficas

- Aplicações para bases de dados

- Multimédia

- Internet protocol / Web

- Robótica & inteligência artificial

**Dados, ou “objetos”**

- Objeto – no contexto de Python, esta designação é aplicada a qualquer dado que possa ser armazenado numa variável, ou passado como parâmetro a uma função

- Cada objeto é caracterizado por: identidade ou referência (identifica a posição da memória onde está armazenado), tipo e valor

- Alguns tipos de objetos podem ter atributos e métodos

- Alguns tipos (classes) de objetos podem ter sub-tipos (sub-classes)

**Sequências de dados**

- Cadeias de caracteres (str)

- Tuplos (tuple) – agregados ou composições de cários 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) | ( )

- 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 vetores na programação imperativa

- é possível modificar elementos individuais das listas

- Os elementos são separados por vírgulas (,) e delimitados por parênteses retos

- Exemplos: [1,2,’a’9] | [(“maria”,33),(“josé”,40)] | [‘lisboa’,[7,“colinas”]] | [ ]

**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

**Acesso a sequências**

- 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

**Instrução de atribuição**

- A instrução de atribuição, em vez de copiar valores, limita-se a associar um dado identificador a um dado objeto

- Assim, a atribuição de uma variável x a uma variável y apenas tem como resultado associar y ao mesmo objeto ao qual x já estava associada

- No caso de objetos mutáveis, há que ter cuidado com efeitos como este: a=[1,2,3] | b=a | b[1:2] = [] | a -> [1,3]

**Funções recursivas**

# 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])

1 + comprimento([2,3])

=

1 +( 1 + comprimento([3]))

=

1 + (1 + (1 + comprimento([]))

=

1 + (1 + (1 + 0))

=

1 + (1 + 1 )

=

1 + 2

=

3

# 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**

- São expressões cujo valor é uma função

- São um “ingrediente” clássico da programação funcional

- Exemplos:

1. lambda x : x+1
   * Função que dado um valor x, devolve x+1
2. 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
3. (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

- Como qualquer objeto, 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: 8\*7=56

- 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

- As expressões lambda também são 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()

- Em Python3, esta função retorna um iterador que pode ser convertido para lista

**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()

- Em Python3, esta função retorna um iterador que pode ser convertido para lista

**Reduzir uma lista a um valor**

- Muitos procedimentos que atuam 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:]).

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

neutro = 0

f = lambda x,s : x+s

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 respetiva soma

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

- Resultado: 15

- Corresponde à função pré-definida reduce()

- Em Python3, esta função está na biblioteca funtools

**Listas de compreensão**

- Mecanismo compacto para processas 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

- Sintaxe (caso simples):

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

- 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]

- 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]

- 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]

- 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 objetos

- Uma classe define um conjunto de objetos caracterizados por diversos atributos e métodos

- É possível definir hierarquias de classes com herança

- As classes surgem na linguagem Python com pouca sintaxe adicional

- Sintaxe:

class <nome-classe>:

<declaração-1>

…

<declaração-N>

- Exemplo

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

def dizer\_ola(self): Exemplo de definição de um

print “Ola” método

- Utilização

>>> x = UmTeste() Criação de uma instância e atribuição a uma variável

>>> x.dizer\_ola() Invocação do método

Ola

**Classes com construtor**

- Exemplo

class Complexo:

def \_\_init\_\_(self,real,imag): O construtor é o método que inicializa um objeto 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

self.r = real

self.i = imag

- Utilização

>>> c = Complexo(-1.5,13.1) Criação de uma instância e atribuição a uma variável

>>> c.r,c.i

(-1.5,13.1)

**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 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 programados para representar

- soma de expressões

- produtos de expressões

- constantes

- ocorrências da variável

- 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:

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, 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\_\_() – implementa a conversão para cadeia de caracteres; suporta a função de conversão str()

- \_\_repr\_\_() – define a representação em cadeia de caracteres que aparece na consola do interpretados; suporta a função repr()

- Atributos

- \_\_class\_\_ - identifica a classe de um dado objeto

- 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)

- …

IA – POWERPOINT 4 – RESUMO

**Definição de Inteligência**

- Segundo [www.dictionary.com](http://www.dictionary.com), “inteligência” é:

- Capacidade de adquirir e aplicar conhecimento

- Capacidade de pensar e raciocinar

- O conjunto de capacidades superiores da mente

**Definição de Inteligência Artificial**

- “Inteligência Artificial” é a disciplina que estuda as teorias e técnicas necessárias ao desenvolvimento de “artefactos” inteligentes

**Tópicos de Inteligência Artificial**

- Agentes

- Noção de agente

- Objetivo da Inteligência Artificial

- Agentes reativos e deliberativos

- Propriedades do mundo de um agente

- Arquiteturas de agentes

- Representação do conhecimento

- Técnicas de resolução de problemas

**Definição de “Agente”**

- Nesta disciplina estudamos técnicas úteis no desenvolvimento de “agentes inteligentes”

- Um “agente” pode ser:

- Entidade com poder ou autoridade de agira

- Entidade que atua em representação de outrem

- Agente – uma entidade com capacidade de obter informação sobre o seu ambiente (através de “sensores”) e de executar ações em função dessa informação (através de “atuadores”)

- Exemplos:

- Agente físico: robô anfitrião

- Agente de software: agente móvel de pesquisa de informação na internet

Uma imagem com texto, diagrama, Tipo de letra, file

Descrição gerada automaticamente

**Agir como o ser humano – o Teste de Turing**

- “Comportamento inteligente” – a capacidade de um artefacto obter desempenho comparável ao desempenho humano em todas as atividades cognitivas.

- Teste de Turing – é uma definição operacional de comportamento inteligente de nível humano:

- Consiste em submeter o artefacto a um interrogatório realizado por um ser humano através de um terminal de texto.

- Se o humano não conseguir concluir se está a interrogar um artefacto ou outro ser humano, então, esse artefacto é inteligente.

- Os sistemas deste tipo serão o objetivo principal da “Inteligência Artificial”?

**A “sala chinesa” de Searle**

- Um humano, que apenas fala uma língua ocidental, documentado com um conjunto de regras escritas num livro nessa língua, e dispondo de folhas de papel, está fechado numa sala.

- Através de uma abertura na sala, o humano recebe folhas de papel com símbolos indecifráveis.

- De acordo com as regras, e em função do que recebe, o humano escreve outros símbolos (que igualmente desconhece) nas folhas brancas e envia-as para o exterior da sala.

- No exterior, no entanto, o que se observa é folhas de papel com mensagens escritas em caracteres chineses a serem introduzidas na sala e respostas inteligentes a essas mensagens a serem devolvidas do interior da sala.

**O argumento de Searle**

- O humano não percebe chinês

- A sala não percebe chinês

- O livro de regras e as folhas de papel também não percebem chinês

- Logo, não há qualquer compreensão de chinês naquela sala

- No entanto, podemos contra-argumentar: embora individualmente, os componentes do sistema (a sala, o humano, o livro, as folhas de papel) não compreendam chinês, o sistema no seu conjunto compreende chinês.

**Tipos de arquiteturas de agente**

- Tipos de agentes

- Reativos simples

- Reativos com estado

- Deliberativos orientados por objetivos

- Deliberativos orientados por funções de utilidades

- Arquiteturas

- Subsunção

- Três torres

- Três camadas

- CARL

**Agente reativo: simples**

Uma imagem com texto, captura de ecrã, Tipo de letra, diagrama

Descrição gerada automaticamente

- O conceito de “regra de condição-ação” é também conhecido como “regra de situação-ação” ou “regra de produção”

- Os agentes ou sistemas reativos simples são também conhecidos como “sistemas de estímulo-resposta” ou “sistemas de produção”

Uma imagem com texto, diagrama, captura de ecrã, Esquema

Descrição gerada automaticamente

**Agente reativo: com estado interno**

**Uma imagem com texto, captura de ecrã, diagrama, Tipo de letra

Descrição gerada automaticamente**

**Uma imagem com texto, diagrama, captura de ecrã, file

Descrição gerada automaticamente**

**Sistemas de Quadro Preto**

- Podem ser vistos como uma elaboração dos sistemas reativos com estado interno.

- Uma “doente de conhecimento” (FC) é um programa que vai fazendo alterações no Quadro Preto.

- Uma FC pode ser vista como um especialista num dado domínio.

- Tipicamente, cada FC rege-se por um conjunto de regras de situação-ação.

**Agente deliberativo: orientado por objetivos**

Uma imagem com texto, captura de ecrã, diagrama, Tipo de letra

Descrição gerada automaticamente

**Agente deliberativo: orientado por função de utilidade**

Uma imagem com texto, captura de ecrã, Tipo de letra, diagrama

Descrição gerada automaticamente

**Propriedades do mundo de um agente**

- Acessibilidade – o mundo é “acessível” se os sensores do agente permitem obter uma descrição completa do estado do mundo; o mundo será “efetivamente acessível” se é possível obter toda a informação relevante ao processo de escolha das ações.

- Determinismo – o mundo é “determinístico” se o estado resultante da execução de uma ação é totalmente determinado pelo estado atual e pelos efeitos esperados da ação.

- Mundo episódico – no caso em que cada episódio de perceção-ação é totalmente independente dos outros.

- Dinamismo – o mundo é “dinâmico” se o seu estado pode mudar enquanto o agente delibera; caso contrário, o mundo diz-se “estático”.

- Continuidade – o mundo é “contínuo” quando a evolução do estado do mundo é um processo continuo ou sem saltos; caso contrário o mundo diz-se “discreto”.

**Mundo de um agente - Exemplos**

Uma imagem com texto, captura de ecrã, número, Paralelo

Descrição gerada automaticamente

**Arquiteturas de agentes: Subsunção**

**Uma imagem com texto, captura de ecrã, Tipo de letra, file

Descrição gerada automaticamente**

- A arquitetura de subsunção procura estabelecer a ligação entre perceção e ação a vários níveis – daqui resulta uma organização em camadas.

- A camada mais baixa é a mais reativa

- O peso da componente deliberativa aumenta à medida que se sobre na estrutura de camadas.

**Arquitetura de Agentes: Três Torres**

**Uma imagem com texto, diagrama, file, Esquema

Descrição gerada automaticamente**

**Arquiteturas de Agentes: Três Camadas**

**Uma imagem com texto, Tipo de letra, captura de ecrã, file

Descrição gerada automaticamente**

**Arquiteturas de Agentes: CARL**

**Uma imagem com texto, diagrama, file, esboço

Descrição gerada automaticamente**

**Tópicos de Inteligência Artificial**

- Agentes

- Noção de agente

- Objetivo da Inteligência Artificial

- Agentes reativos e deliberativos

- Propriedades do mundo de um agente

- Arquiteturas de agentes

- Representação do conhecimento

- Técnicas de resolução de problemas

**Representação do conhecimento**

- Redes semânticas

- Redes semânticas genéricas

- Sistema de “frames”

- Herança e raciocínio não-monotónico

- Relação com diagramas UML

- Exemplo para aulas práticas

- Lógica proposicional e lógica de primeira ordem

- Linguagem KIF

- Engenharia do conhecimento

- Ontologia geral

- Redes de Bayes

**Redes Semânticas**

- Redes semânticas são representações gráficas do conhecimento

- Têm a vantagem da legibilidade

- As redes semânticas podem ser tão expressivas quanto a lógica de primeira ordem

pp 34