Advanced Institute for Artificial Intelligence

Python

Advanced Institute for Artificial Intelligence

https://advancedinstitute.ai

Sumario

- Introdução
- Estruturas e Função de Controle
- Coleções
- Programação Orientada a Objetos
- Manipulação de arquivos
- Processos e Threading

Introdução

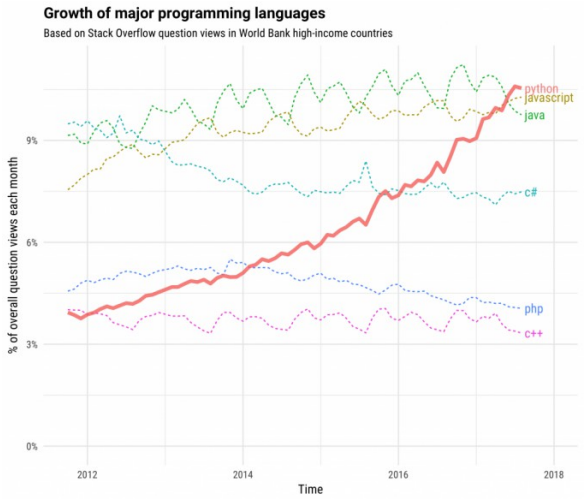
Python

- Python é uma linguagem interpretada

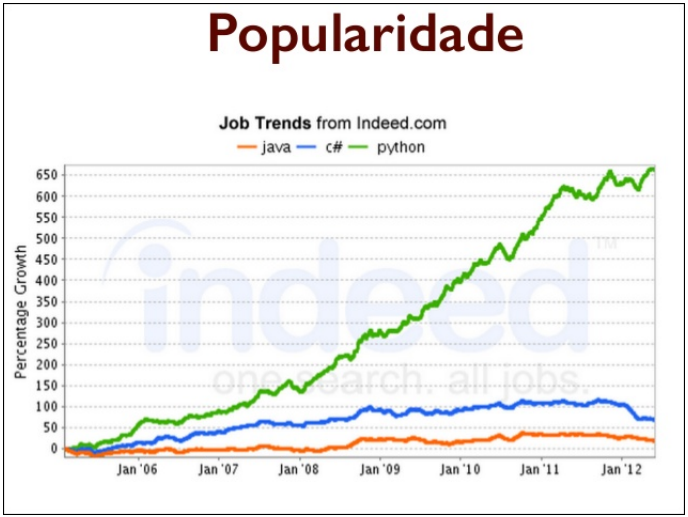
Interpretada x Compilada

- **Interpretada**: Python, Perl, Lua
- **Compilada**: C, Fortran, C

- Criada nos anos 90, se tornou extremamente popular nos últimos anos.



Popularidade



Introdução

- Caminho do Python no Sistema

```
1 which python
```

- Versão do Python

```
1 python -V
```

Usando Python

- Iniciando interpretador Python

```
1 python
```

- Python 3.6.8 —Anaconda, Inc.— (default, Dec 30 2018, 01:22:34)
- [GCC 7.3.0] on linux
- Type "help", "copyright", "credits" or "license" for more information.
- >>> Esse é o prompt para receber comandos python

- Ctrl+D sai do interpretador

- O Python tem duas versões principais: 2 e 3
- Sempre que possível deve-se utilizar Python 3
- Entretando, alguns códigos e bibliotecas só são compatíveis com Python 2, portanto pode ser que tenhamos que utilizá-lo algumas vezes.

Usando Python

Comando print

- Python 2

```
1 print "hello world"
```

- Python 3

```
1 print ("hello world")
```

- print "hello world"

Comentários no código

- # : comentando uma linha
- "' : começar e terminar bloco de comentário
- """ : começar e terminar bloco de comentário

Indentação

- O controle de início e fim de blocos de código é feito por meio de Indentação
- Indentação pode ser controlada por um tamanho fixo de espaços em branco
- Exemplo

```
1 print ("teste")
2 if (i == 0):
3     print ("0")
4 else:
5     print ("outro valor")
6     if (i >= 0):
7         print (">=0")
```

Tipos de dados - Números

Existem três tipos numéricos em python: números inteiros, números de ponto flutuante e números complexos.

- Booleanos são um subtipo de números inteiros.
- Inteiros têm precisão ilimitada.
- Números de ponto flutuante são geralmente implementados usando tipo Double em C

Operações com Números

- =: Sempre usado para **atribuições**
- ==, !=, <>, >, <, >=, <=: Comparações

```
1 a = 10
2 b = 5
3 print(a == b)
4 print(a != b)
5 print(a <> b)
6 print(a > b)
7 print(a < b)
8 print(a >= b)
9 print(a <= b)
```

Tipos de dados - Strings

Strings podem ser manipuladas de diversas maneiras em Python

- podem ser representadas usando aspas simples ' ' ou aspas duplas " "
- É possível utilizar catacteres escape

Funções

- A palavra-chave def é usada para definir funções
- Deve ser definida antes de ser utilizada
- O valor de retorno padrão é None

Função

Argumento pode ser gerado da seguinte forma:

- nome de variável
- nome de variável e valor padrão

Escopo de variável

- variáveis possuem escopo local ao bloco onde são criadas
- Pode ser definidas variáveis globais

Funções

Função sem argumentos:

```
1  def greeting():
2      print("hello world")
3
4  greeting()
```

Argumento de Função

```
1  def numsquare(num):
2      return num * num
3
4  number=10
5
6  numsquare(number)
7
8  def numsquare(num=10):
9      return num * num
10
11 numsquare()
```

Obtendo dados do usuário

A função input() é utilizada para aguardar um valor digitado no terminal pelo usuário

```
usrip = input("numero inteiro: ")
usrnum = int(usrip)
sqrnum = numsquare(usrnum)
print("O quadrado do numero eh: {}".format(sqrnum))

usrip = input("float: ")
usrnum = float(usrip)
sqrnum = numsquare(usrnum)
print("O quadrado do numero eh: {}".format(sqrnum))

usrname = input("nome: ")
print("nome: ",usrname)
```

Usando bibliotecas adicionais

A palavra reservada import permite adicionar pacotes que não são nativos do Python

```
1  import subprocess
2
3  # Executa um comando linux no terminal
4
5  subprocess.call('date')
```

A palavra reservada from permite importar apenas parte de um pacote

```
1  from sklearn.model_selection import train_test_split
```

Estruturas e Função de Controle

As funções padrão de controle de fluxo de execução estão disponíveis no Python:

- if
- for
- while

if

- As instruções if avaliam uma condição, caso seja verdadeira executa o bloco seguinte
- Pode ser combinado com uma estrutura else, que é executada quando a condição não é verdadeira no bloco if

Exemplo:

```
1  var = 100
2
3  if (var==100):
4      print("100")
5  else:
6      print("not 100")
```

for

- executam um certo bloco de código para um número conhecido de iterações.
- Um bloco de código pode ser executado para o número de itens existentes em uma lista, dicionário, variável de sequência ou tupla
- Um bloco de código pode ser executado em um intervalo contado de etapas

Exemplo

```
1  a=(10,20,30,40,50)
2  for b in a:
3      print ("square of " + str(b) + " is " +str(b*b))
```

while

- O loop while é executado enquanto uma declaração condicional retorna true
- A instrução condicional é avaliada toda vez que um bloco de código é executado
- A execução para no momento em que a instrução condicional retorna false.

Exemplo:

```
1 count = 0
2 while (count $<$ 9):
3     print("itera   o",count)
4     count+=1
```

Operadores Lógicos

Principais Operadores Lógicos

- *and*
- *or*
- *not*

```
1 x = True
2 y = False
3
4 print('x and y is',x and y)
5 print('x or y is',x or y)
6 print('not x is',not x)
```

x and y is False
x or y is True
not x is False

Coleções

- Coleções são implementações de estrutura de dados
- Permitem guardar e manipular conjuntos de valores de maneira organizada e otimizada

```
1 friends = [ 'Joseph', 'Glenn', 'Sally' ]
2 carryon = [ 'socks', 'shirt', 'perfume' ]
```

Coleções em python:

- list: Lista
- set: Conjunto
- dictionary: Dicionárop / Hash Table
- tuple: Tupla
- entre outros....

List

Os elementos na lista (list) são separados por vírgulas.
Um elemento da lista pode ser qualquer objeto Python - até outra lista
Uma lista pode estar vazia
Operador index representa uma posição na lista

list = [1, 24, 76]
list = ['red', 'yellow', 'blue']
list = ['red', 24, 98.599999999999994]
list = [1, [5, 6], 7]
l[0] => 1

List

- Listas são mutáveis

```
1 lotto = [2, 14, 26, 41, 63]
2 print(lotto)
```

[2, 14, 26, 41, 63]

```
1 lotto[2] = 28
2 print(lotto)
```

[2, 14, 28, 41, 63]

- Operador **len** retorna tamanho da lista

```
1 print (len(lotto))
```

5

Operações

- **append**: Adiciona elementos no fim da lista
- **in**: pode ser usado para verificar se um elemento existe na lista
- **sort**: Ordena a lista
- **split**: Quebra uma string em partes menores usando estrutura de lista

Dicionários

Dicionários são uma implementação de hashtable

- Mapeia "chaves" para valores

Operações:

- print, del, len, in

Métodos:

- keys(), values(), items()

Dicionários

```
1  eng2sp = {}
2  eng2sp['one'] = 'uno'
3  eng2sp['two'] = 'dos'
4
5  eng2sp = { 'one': 'uno', 'two': 'dos', 'three':'tres' }
```

List Comprehensions

Aplica uma expressão a cada elemento da lista

```
1  vec = [2, 4, 6]
2  [3*x for x in vec]
```

[6, 12, 18]

```
1  [3*x for x in vec if x > 3]
```

[12, 18]

Slicing

- Listas podem ser filtradas por meio de 'slicing'
- Formato para realizar 'slicing' em uma lista:

```
1  s[start:end:step]
```

Elementos:

- **s**: um objeto que pode ser manipulado por 'slicing'
- **start**: primeiro índice para iniciar a iteração
- **end**: último indíce, NOTE que o índice final não será incluído na fatia resultante
- **step**: escolha o elemento a cada índice de etapa

Slicing

Alguns Exemplos:

- Selecionar itens a partir do índice start até stop-1

```
a[start:stop]
```

- Selecionar itens a partir do índice start até o final

```
a[start:]
```

- Selecionar itens a partir do início start até stop-1

```
a[:stop]
```

- Selecionar itens a partir do início até o final

```
a[:]
```

Slicing

Alguns Exemplos:

- Selecionar itens a partir do índice start não passando de stop-1, realizando pulos definidos na variável step

```
a[start:stop:step]
```

- Último item da lista

```
a[-1]
```

- Últimos dois itens da lista

```
a[-2:]
```

- Tudo menos os dois últimos

```
a[:-2]
```

Slicing

Alguns Exemplos:

- Quando a lista possuir mais de uma dimensão, é necessário realizar o slicing separadamente em cada dimensão

- Apagando elementos de uma lista

```
1  del a[3:7]
```

Manipulação de Texto

Abrir um arquivo:

- Preparar o arquivo para leitura:
- Vincula a variável do arquivo ao arquivo físico
- Posiciona o ponteiro do arquivo no início do arquivo.

Formato:

```
1   <variavel do arquivo> = open (<nome do arquivo>, "r")
```

Exemplo:

```
1  inputFile = open ("data.txt", "r")
2
3  filename = input ("Digite o nome do arquivo de entrada:")
4
5  inputFile = open (filename, "r")
```

Comando para fechar um arquivo Formato:

```
1 <name of file variable>.close()
```

Exemplo:

```
1 inputFile.close()
```

Manipulação de Texto

- Normalmente, a leitura é feita dentro do corpo de um loop
- Cada execução do loop lê uma linha do arquivo em uma string

```
1 for <variavel para armazenar uma sequencia> em <nome da variavel do
      arquivo>:
2
3       <Faca algo com a string lida no arquivo>
```

Exemplo:

```
1 for line in inputFile:
2       print (line)
```

Advanced Institute for Artificial Intelligence

Programação Orientada a Objetos

Advanced Institute for Artificial Intelligence

https://advancedinstitute.ai

Orientação a objeto

Orientação a Objetos surgiu da necessidade de modelar sistemas complexos

- Modelar problemas utilizando um conjunto de componentes autocontidos, e integráveis
- Determinar como um objeto deve se comportar e como deve interagir com outros objetos

Algumas iniciativas:

- Simula 67 (60)
- Smalltalk (70)
- C++ (80)

Orientação a objeto

Conceitos essenciais:

- Classes e objetos
- Atributos e Métodos
- Herança
- Encapsulamento

Orientação a objeto

Os objetos reais possuem duas caracterísicas:

- Atributos (Estado)
- Comportamento

Exemplos:

- cachorro
  - Atributos: nome, cor, raça
  - Comportamento: latindo, abanando o rabo, comendo

Orientação a objeto

Um objeto de software é conceitualmente similar aos objetos reais

- Objetos armazenam seu estado em atributos
  - Correspondentes às variáveis em programação estruturada.
- Objetos expõem seu comportamento através de métodos
  - Correspondentes às funções em programação estruturada.

Orientação a objeto

Exemplos de objeto:

- Gerenciador de Dados de Alunos
  - Atributos: lista de alunos
  - Comportamentos: filtrar alunos por nome, incluir aluno, alterar aluno
- Biblioteca Matemática
  - Atributos: Matriz
  - Comportamentos: calcular transposta, multiplicar, somar

Orientação a objeto

Empacotar o código em objetos individuais fornece:

- Modularidade
  - Objetos são independente
- Encapsulamento
  - Os detalhes da implementação de um objeto permanecem ocultos
- Reuso
  - Objetos podem ser reutilizados em diferentes programas
- Fraco acoplamento
  - Objetos podem ser substituídos facilmente

Orientação a objeto

Uma classe é o projeto a partir do qual objetos individuais são criados

- Ela define os atributos e os métodos correspondentes aos seus objetos.
- Outros possíveis membros de uma classe são:
  - Construtores: define as operações a serem realizadas quando um objeto é instanciado.
  - Destrutores: define as operações a serem realizadas quando um objeto é destruído.

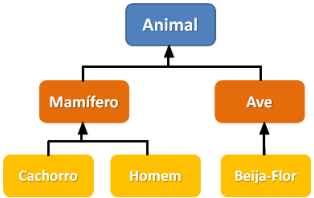
Orientação a objeto

Outras características de uma classe:

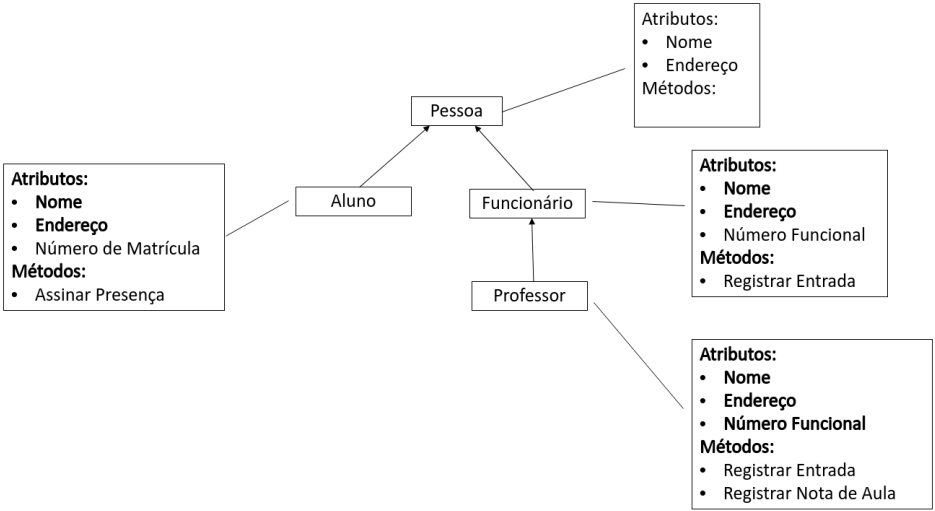
- Uma classe pode herdar características de outra classe e incluir novas características
- Atributos de uma classe podem ser protegidos, sendo possível alterar seu conteúdo por meio apenas de métodos da própria classe
- Métodos podem ser reescritos

Orientação a objeto

- O relacionamento de Herança define um relacionamento do tipo generalização
- Indica que uma classe (subclasse) é especializada para gerar uma nova (superclasse)
- Tudo que a superclasse possui, a subclasse também vai possuir
- Em Python, todas as classes herdam a classe Object



Orientação a objeto



Método Construtor em Python

```
1 def __init__(self):
2     Comandos do construtor
```

Parâmetro para referenciar ao objeto criado: self

- Para acessar e criar atributos em um objeto, o caracter "." deve ser usado após o nome do objeto.

```
1 class Critter:
2     def __init__(self, name):
3         self.name = name
```

Atributos e Métodos de um objeto são acessíveis através de "."

```
1  class Classe():
2      def __init__(self):
3          self.atributo = 0
4      def metodo(self):
5          print(self.atributo)
6  obj = Classe()
7  print(obj.atributo)
8  obj.metodo()
```

A Convenção para definir Métodos ou variáveis privados em Python é colocar como prefixo de seu nome "__"

PS: Não existem elementos privados "verdadeiros" em Python. É possível acessar qualquer método/atributo de uma classe.

- Exemplo:

```
__a
__my_variable
```

Heranças são definidas na declaração da classe, logo após seu nome

```
class teste(object):
    def __init__(self, X):
        self.X = X
```

Orientação a objeto

Exemplo de uma classe em Python

```
1  class MyClass:
2      def function(self):
3          print("This is a message inside the class.")
```

Instanciando um objeto e chamando métodos:

```
1  myobjectx = MyClass()
2  myobjectx.function()
```

Exemplo de uma classe em Python

```
1  # Classe que representa uma coordenada X Y
2  class Coordinate(object):
3      #define um construtor
4      def __init__(self, x, y):
5          # configura coordenada x e y
6          self.x = x
7          self.y = y
8      #reimplementa a fun   o __str__
9      def __str__(self):
10         # Representa  o em string da coordenada
11         return "<" + str(self.x) + "," + str(self.y) + ">"
```

Orientação a objeto

```
def distance(self, other):
    # Calcula distancia euclidiana entre dois pontos
    x_diff_sq = (self.x-other.x)**2
    y_diff_sq = (self.y-other.y)**2
    return (x_diff_sq + y_diff_sq)**0.5
```

Teste de Uso da Classe

```
c = Coordinate(3,4)
origin = Coordinate(0,0)
print("Coordenada 1:")
print(c)
print(c.distance(origin))
```

Teste com atributos protegidos

```
1  class MyClass:
2      __variable = 0
3      def setvariable(self,newvar):
4          self.__variable = newvar
5      def getvariable(self):
6          return (self.__variable)
7      def function(self):
8          print("This is a message inside the class.")
```

Teste com atributos protegidos

```
1  var="rs2"
2  myobjectx = MyClass()
3  myobjectx.function()
4  print(myobjectx.getvariable())
5  var="rs3"
6  myobjectx.setvariable(var)
7  print(myobjectx.getvariable())
```

Orientação a objeto

Subclasses conseguem acessar métodos de suas superclasses. O método *super* acessa o construtor da classe mãe

```
1  class Mae()
2      def __init__(self):
3          self.a = 0
4      def print_a(self):
5          print(self.a)
6  class Filha(Mae):
7      def __init__(self):
8          super().__init__()
9          self.b = 1
10     def print_b(self):
11         self.print_a()
12         print(self.b)
13 obj = Filha()
14 obj.print_b()
```

- A orientação a objetos permite construir códigos facilmente reusáveis
- Além disso, o código fica mais legível e fácil de dar manutenção