Advanced Institute for Artificial Intelligence

Python

Advanced Institute for Artificial Intelligence

https://advancedinstitute.ai

Sumario

Introdução

- Python é uma linguagem interpretada
- Caminho do Python no Sistema
  - which python
- Versão do Python
  - python -V

Usando Python

- Iniciando interpretador Python
  - 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

Usando Python

Comando print

- print "hello world"
- Em python 2 é possível utilizar dessa forma:
- print "hello world"
- Em python 3 é obrigatório utilizar ( )
- print ("hello world")

Comentários no código

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

Usando Python

Indentação

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

```
print ("teste")
if (i == 0):
        print ("0")
else:
        print ("outro valor")
        if (i >= 0):
                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

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 tipo padrão

Escopo de variável

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

Funções

Função sem argumentos:

```
def greeting():
    print("hello world")


greeting()
```

Argumento de Função

```
def numsquare(num):
        return num * num

number=10
numsquare(number)

def numsquare(num=10):
        return num * num

numsquare()
```

Obtendo dados do usuário

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

```
usrip = input("número inteiro: ")
usrnum = int(usrip)
sqrnum = numsquare(usrnum)
print("Square of entered number is: ".format(sqrnum))

usrip = input("float: ")
usrnum = float(usrip)
sqrnum = numsquare(usrnum)
print("Square of entered number is: ".format(sqrnum))

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

Usando bibliotecas adicionais

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

```
import subprocess
# Executa um comando linux no terminal
subprocess.call('date')
```

A palavra reservada from premite importar apenas parte de um pacote

exemplo:

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

Estruturas e Função de Controle

Argumento pode ser gerado da seguinte forma:

- if
- for
- while

Estruturas e Função de Controle

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:

```
var = 100
if (var==100):
        print("100")
else:           print("not 100")
```

Estruturas e Função de Controle

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:

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

Estruturas e Função de Controle

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:

```
count = 0
while (count < 9):

print("iteração",count)
count+=1
```

Coleções

Uma coleção nos permite colocar muitos valores em uma única "variável"

Uma coleção é útil para transportar valores em um único pacote.

friends = ['Joseph', 'Glenn', 'Sally']

carryon = ['socks', 'shirt', 'perfume']

Coleções em python:

list, set, stack, dictionary, tuplas, 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

lotto = [2, 14, 26, 41, 63]

print lotto

[2, 14, 26, 41, 63]

lotto[2] = 28

print lotto

[2, 14, 28, 41, 63]

Operador **len** retorna tamanho da lista

print len(lotto)

5

List

Append adiciona elementos no fim da lista

Operador **in** pode ser usado para verificar se um elemento existe na lista

sort classifica a lista

split quebra uma string em partes menores usando estrutura de lista

Dicionários

Dicionários são estruturas que mapeiam chaves para valores

Operações:

print, del, len, in

Métodos:

keys(), values(), items()

Dicionários

eng2sp =

Adicionando valores

>>> eng2sp['one'] = 'uno'

>>> eng2sp['two'] = 'dos'

declarando dicionário com valores iniciais

>>> eng2sp = [] 'one': 'uno', 'two': 'dos', 'three':'tres']

List Comprehensions

Aplica uma expressão a cada elemento da lista

vec = [2, 4, 6]

>>> [3*x for x in vec]

[6, 12, 18]

>>> [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: 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]
- Copia toda a lista
  - 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 úlitmos
  - 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
  - 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: <variável do arquivo> = open (<nome do arquivo>, "r")

Exemplo: inputFile = open ("data.txt", "r")

filename = input ("Digite o nome do arquivo de entrada:")

inputFile = open (filename, "r")

Manipulação de Texto

Comando para fechar um arquivo

Formato:

<name of file variable>.close()

Exemplo:

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

Formato:

for <variável para armazenar uma sequência> em <nome da variável do arquivo>:

<Faça algo com a string lida no arquivo>

Exemplo:

for line in inputFile: print (line)

Orientação a objeto

Orientação a Objetos surgiu da necessidade de modelar sistemas complexos

- Modelar problemas utilizando um conjunto de componentes autocontindo, 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:

- Estado (Atributos)
- Comportamento

Exemplos:

- cachorro
  - Estado: 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
  - Estado: lista de alunos
  - Comportamento: filtrar alunos por nome, incluir aluno, alterar aluno
- Biblioteca Matemática
  - Estado: Matriz
  - Comportamento: 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 criado.
  - Destrutores: define as operações a serem realizadas quando um objeto é destruído.

Orientação a objeto

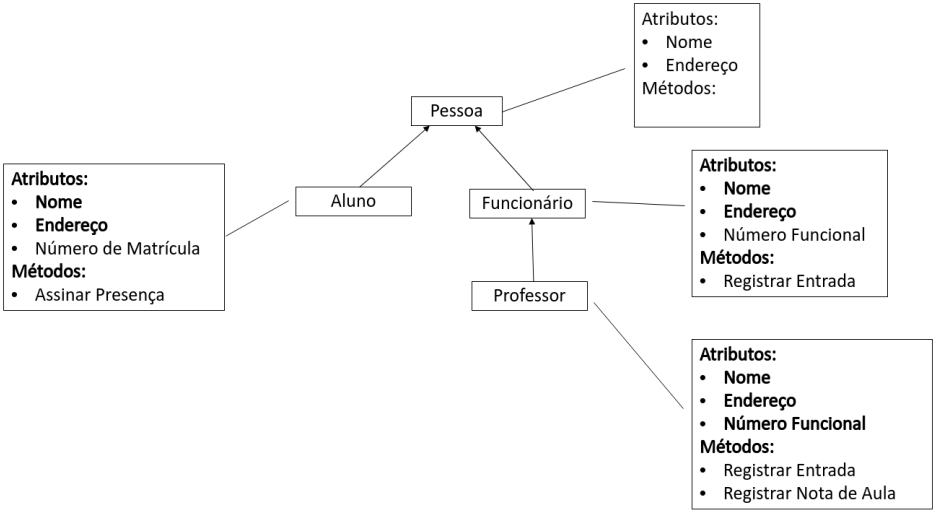
Outras características de uma classe:

- 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



Orientação a objeto

Método Construtor e Destrutor em Python

```
def __init__(self):
    Comandos do construtor
def __del__(self):
    Comandos do destrutor
```

Parâmetro para referenciar ao objeto criado: self

self.name cria uma variável name e associa ao objeto criado.

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

Orientação a objeto

Para definir Métodos Privados em Python é necessário incluir:

__

Exemplo:

```
__a
__my_variable
```

Para definir que Classe deve herdar outra Classe, deve se colocar o nome da classe a ser herdade entre (), logo após o nome da classe:

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

Orientação a objeto

Exemplo de uma classe em Python

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

Instanciando um objeto e chamando métodos:

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

Orientação a objeto

Exemplo de uma classe em Python

```
# Classe que representa uma coordenada X Y
class Coordinate(object):
    #define um construtor
    def __init__(self, x, y):
        # configura coordenada x e y
        self.x = x
        self.y = y
    #reimplementa a função __str__
    def __str__(self):
        # Representação em string da coordenada
        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

```
class MyClass:
    __variable = 0
    def setvariable(self,newvar):
        self.__variable = newvar
    def getvariable(self):
        return (self.__variable)
    def function(self):
        print("This is a message inside the class.")
```

Teste com atributos protegidos

```
var="rs2"
myobjectx = MyClass()
myobjectx.function()
print(myobjectx.getvariable())
var="rs3"
myobjectx.setvariable(var)
print(myobjectx.getvariable())
```

Funções Lambda

- São funções anônimas criadas com a palavra-chave lambda
- São restritas a uma única linha
- Possuem um return implícito

Sintaxe

- Iniciamos com a palavras lambda
- Em seguida são definidos parâmetros
- Após os parâmetros colocamos :
- Após : vem a lógica
- Opcionalmente podemos ter variáveis

Exemplo de função lambda

```
(lambda x: x + 1)(2)

x = lambda a, b : a * b
print(x(5, 6))
```

Map

Map é uma função para derivar uma lista a partir de uma existente

```
map(object, iterable_1, iterable_2, ...)
```

O iterador da função map () pode ser um dicionário, uma lista etc. A função map () basicamente mapeia todos os itens da entrada iteráveis, para o item correspondente no iterador de saída, de acordo com a lógica definida pela função lambda.

```
#mapeia os itens de entrada da lista para uma lista de saída que registra
#o resto da divisão de cada item por 2
numbers_list = [2, 6, 8, 10, 11, 4, 12, 7, 13, 17, 0, 3, 21]
mapped_list = list(map(lambda num: num % 2, numbers_list))
```

Filter

Filtros são utilizados para gerar uma nova lista com elementos que atendam uma certa condição

- Cada item da lista é avaliado como True ou False de acordo com uma função lambda
- Apenas os elementos que retornam True da função lambda são retornados

```
#filtra elementos maiores que 7
numbers_list = [2, 6, 8, 10, 11, 4, 12, 7, 13, 17, 0, 3, 21]
filtered_list = list(filter(lambda num: (num > 7), numbers_list))
print(filtered_list)
```

Reduce

Reduce é uma função para executar operações em todos os elementos de uma lista e retornar um resultado.

Por exemplo, se você deseja calcular o produto de uma lista de números inteiros.

```
product = 1
list = [1, 2, 3, 4]
for num in list:
    product = product * num
print(product)
```

Usando reduce e lambda

```
import functools
product = functools.reduce((lambda x, y: x * y), [1, 2, 3, 4])
print(product)
```

Enumerate

Enumerate percorre uma lista e retornando o índice de cada valor

```
L = [’maça’, ’banana’, ’laranja’]
for idx, val in enumerate(L):
  print("index  %d  value  %s" % (idx, val))
```

Organização de Código

- Separar o código em partes menores é fundamental para obter reuso
- Uma estratégia de reusar código é utilizar módulos
- Módulos sao arquivos com fragmentos de código que podem ser importados

Exemplo de uso de Módulo

Arquivo utils/lib.py

```
def printmsg():
    print("mensagem da lib")
```

Arquivo programaPrincipal.py

```
import sys
sys.path.append('utils')

from lib import *

print("programa principal")
printmsg()
```

Organização de Código

A utilização de módulos permite separar o código em diferentes arquivos

- O nome do módulo é igual ao nome do arquivo
- É possível importar o módulo completo ou parte do módulo apenas
- É possível criar módulos dentro de módulos, utilizando sub-diretório
- Ao importar um módulo é possível consulta o seu conteúdo usando a função dir, que retorna os recursos disponíveis em um módulo

Organização de Código

Distribuindo módulos

- A utilização de módulos é possível se os arquivos estiverem disponíveis numa certa estrutura
- Para recriar um ambiente desse tipo em outro computador é necessário construir a estrutura da mesma maneira
- Para distribuição ampla de um programa essa estratégia não é intuitiva

Organização de Código

Packaging

- Um recurso útil para distribuir módulos é utilizar a biblioteca setuptools, que cria pacotes
- Um pacote pode conter diversos módulos, bem como, informações diversas sobre os módulos
  - Versão, autor, localização do código fonte original, entre outras
- Essa ferramenta é útil para montar os módulos de forma automática em um ambiente virtual

Organização de Código

Distribuindo pacotes

- O recurso de criar um pacote é muito útil para distribuir uma aplicação, porém, o usuário realiza a instalação a partir de um conjunto de arquivos
- Toda vez que for necessário usar esses módulos, será necessário seguir um conjunto de procedimentos de instalação
- As ferramentas pip e conda auxiliam nesse cenário permitindo a criação de vários ambientes virtuais, e também, permitindo instalar um pacote com apenas um comando, conhecendo apenas o nome e localização do pacote

Organização de Código

Índice público

- Pypi é um índice público de Módulos
- Qualquer desenvolvedor pode publicar módulos
- Uma vez publicados, a partir de um ambiente conda, outros desenvolvedores podem obter e utilizar

Organização de Código

Instalando a partir de Índices públicos

- Ao instalar de um índice público o próprio ambiente virtual se encarrega de colocar os arquivos nos locais corretos
- Ao criar uma atualização, o ambiente virtual permite fazer uma atualização
- É possível inclusive trocar versões ou eliminar completamente uma das versões

Organização de Código

Usando índice público de teste

- O índice https://test.pypi.org/ permite que usuários utilizem um sistema de índice público de distribuição
- Esse não é o índice oficial do PIP adequado para testar como funciona um índice público
- Para usar esse índice é necessário criar uma conta e ativar o token

Organização de Código

- Usando recurso de packaging é possível distribuir o código de uma forma que outra pessoa possa reconstruir no seu próprio ambiente virtual a partir do fonte
  - Nesse cenário a pessoa obtem o código e utiliza o instalador para organizar o fonte nos diretórios adequados no ambiente virtual
- Quando utilizamos um índice público o pip faz esse papel, e obtem o programa de m repositório comum
  - Índices públicos são apropriados quando se deseja disponibilizar um código amplamente