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 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)

Manipulação de Texto

Escrita de arquivo

Exemplo 1:

f = open("f.txt", "w")

f.write("teste")

f.close()

Exemplo 2:

with open("new$file.txt$", "w")$as f$:

f.write("This is a sample line of text")

f.write("Yet another line")

Orientação a objeto

TUDO EM PYTHON É UM OBJETO (e tem um tipo)

pode criar novos objetos de algum tipo

pode manipular objetos

pode destruir objetos

usar explicitamente del ou apenas “esquecê-los” (Garbage collector)

Orientação a objeto

objetos são uma abstração de dados

possuem uma representação interna

através de atributos de dados

uma interface para interagir com o objeto

através de métodos

define comportamento, mas oculta implementação

Orientação a objeto

Vantagens da orienração a objetos

Aumento de modularidade para dividir a complexidade de implementação

Reuso de código entre diversos programas

baixo acoplamento por meio de interfaces

Orientação a objeto

Definindo uma nova classe

class Coordinate(object):

definição de atributos definição de métodos

Implementação de construtos: método $_{init}$

def $_{init}{}_{(self,x,y):self.x=xself.y=y}$

Orientação a objeto

overload de método print: def $_{s}tr_{(}{}^{self)}$:

Exemplo:

def $_{s}tr_{(}{}^{self)}{:}return"<"+str(self.x)+","+str(self.y)+">"$

MultiThreading

Conflito de Versões

- Conflitos sãoo situações em que o git não consegue atualizar um arquivo com uma nova versão da branch automaticamente
- Nesses casos o git avisa essa situação e pede para que o desenvolvedor resolva os conflitos

Exemplo de conflito:

```
<<<<<<< branch1
#include "test.h"

int main()
{
   return 0;
}

=======
// needed a source code file
```