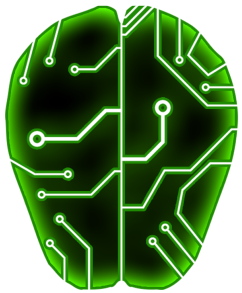
Curso de Python: Parte 3

Conceitos Intermediários



COMPUTER SOCIETY

IFB

Guilherme Paiva

Instituto Federal de Brasília, Campus Taguatinga

Sumário

Estrututas de dados avançadas

- Compreensão de listas
- Dicionários

Listas

Podemos percorrer listas de duas formas:

- Usando o índice:

```
lista = [2,4,6,8]
for i in range(len(lista)):
    print('Elemento: ',lista[i])
    print('Índice: ',i)
```

- acessando diretamente o elemento:

```
lista = [2,4,6,8]
for elem in lista:
    print('Elemento:',elem)
```

Compreensão de Listas

- É uma forma concisa de criar listas.
- O que era:

```
lista = range(1,11)
quadrados_lista = []
for item in lista:
    quadrados_lista.append(item**2)
```

- Se torna:

```
lista = range(1,11)
lista_quadrada = [item**2 for item in lista]
```

Compreensão de Listas

- Também é possível inserir condições ao criar listas.
- Gerando números pares:

```
lista = range(1,11)
lista_de_pares = []
for item in lista:
    if item%2 == 0:
        lista_de_pares.append(item)
```

- Com compreensão de listas:

```
lista = range(1,11)
lista_de_pares = [item for item in lista if item %2 == 0]
```

Estrutura do código

- A estrutura do código deve ser sempre da seguinte forma:

```
lista = [expressao(item) for item in input_lista if condicao]
```

- Note que expressão e condição são atributos opcionais.
- Também é possível adicionar mais de uma expressão.

```
l1 = [10,20,30]
l2 = [40,50,60]
somas = [x+y for x in l1 for y in l2]
```

Dicionários

- São estruturas de dados, que podem ser indexadas por valores imutáveis, tais como strings ou números.
- Um jeito melhor de definir dicionários, é pensando em um modelo chave valor, onde as chaves são únicas.

```
emails = {'Nicolau' : 'nico12@mail.com',
'Geraldo' : 'geraldorivia@velen.com'}
emails['Geraldo']
dic_quad = {x: x**2 for x in (2,4,6)}
```

Lendo múltiplos dados

- Já vimos como ler dados inseridos por um usuário e como converter para o tipo de dado desejado.
- Mas como ler vários dados em sequência, ou atribuir para várias variáveis?
- Podemos usar funções de strings para isso.

Entrada de dados

- Para ler dados separados por um espaço, ou outro delimitador:

```
a = input().split()
```

- Para converter todos em um tipo de dado específico:

```
b = [int(i) for i in input().split()]
```

- Passando valores a diferentes variáveis:

```
idade, altura = [float(i) for i in input("""Digite sua idade
e altura""").split()]
```

Arquivos

- É tratado como um objeto, podendo ser atribuído a variáveis.
- Para passar uma entrada em um arquivo de texto para um programa no terminal basta digitar python3 arquivo.py <arquivo.txt
- Criando um objeto arquivo:

```
file = open(r"nome_do_arq", "modo")
```

Modos de operação

- O modo de abertura de um arquivo deve ser informado na função open.
- Os modos existentes são:
  - 'r' : apenas leitura, padrão
  - 'w' : apenas escrita, o conteúdo é zerado
  - 'r+' : leitura e escrita
  - 'a' : apenas acrescentar
  - 'a+' : acrescentar e ler

Funções de leitura e escrita

- Após abrir um arquivo, é necessário usar funções para manipulá-lo, são elas:
  - Leitura:
    - arq.read()
    - arq.readline()
    - arq.readlines()
  - Escrita:
    - arq.write(str)
    - arq.writelines(list)
  - Função seek(), tell()
  - Função close()

Classes e Objetos

- Nos programas que escrevemos até agora, desenvolvemos nossos programas em torno de funções para manipular dados, isso é chamado de programação procedural.
- Classes e Objetos são dois aspectos principais da programação orientada a objetos.
- Uma **classe** cria um novo tipo, o qual os **objetos** são instâncias dessa classe.
- Uma analogia é a classe int, que significa que objetos desta classe armazenam dados do tipo número inteiros.

Classes e Objetos

- **Objetos** podem armazenar dados usando variáveis normais que **pertencem** ao objeto.
- Variáveis pertencentes a objetos ou classes são geralmente chamadas de **atributos**
- Funções pertencentes a classes são chamadas de **Métodos**
- Esta metodologia é importante para diferenciar funções e variáveis independentes das que pertencem à classes ou objetos.

Self

- **Self** é uma palavra reservada do Python, utilizada em métodos de classes.
- Quando criamos um método dentro de uma classe, é necessário que o self seja o primeiro parâmetro deste método, porém não é necessário passar valor para este parâmetro.
- O python atribui para o self o nome do objeto que está sendo instanciado quando este método for invocado, porém não precisamos nos preocupar com isso.

Primeira Classe

- Para criamos uma classe, basta usarmos a palavra reservada class seguido do nome da classe.
- Por convenção, nome de classes se iniciam com letras maiúsculas.

```
class Robot:
    pass
ed = Robot()
print(ed)
```

Métodos

- Criamos métodos de classes assim como criamos funções independentes, com a diferença do self.

```
class Robot:
    def diz_oi(self):
        print('Olá, humano!')
otis = Robot()
otis.diz_oi()
```

Método init

- O método init é invocado assim que um objeto é instanciado. É util para fazer inicializações com o objeto.

```
class Robot:
    def __init__(self, name):
        self.name = name
    def diz_oi(self):
        print('Olá, humano! Me chamo',self.name)
otis = Robot('Otis')
otis.diz_oi()
```

Variáveis de Classes e Objetos

- As variáveis podem pertencer à classe ou ao objeto.
- **Atributo de Classe**
  - É uma variável compartilhada, há apenas uma cópia, sendo esta visível para todos os objetos desta classe. Caso algum objeto faça uma alteração nesta variável, esta mudança será vista por todos os outros objetos.
- **Atributo de Objetos**
  - São individuais de cada instância. Neste caso, cada objeto possui uma cópia desta variável e não tem nenhuma relação com esta mesma variável de outro objeto.

Variáveis de Classes e Objetos

```
class Robot:
    populacao = 0
    def __init__(self, name):
        self.name = name
        print("*Inicializando {}*".format(self.name))
        Robot.populacao +=1
    def diz_oi(self):
        print('Saudações, humano! Meu criador me chama de',self.name)
```

Variáveis de Classes e Objetos

```
def auto_destuir(self):
    if Robot.populacao == 0:
        print('Os robôs foram extintos')
        return
    else:
        print('{} está se autodestruindo!!'.format(self.name))
        Robot.populacao -=1
    if Robot.populacao == 0:
        print('{} foi o último de sua espécie'.format(self.name))
    else:
        print('Ainda temos {} robôs em funcionamento'.format(
                                                    Robot.populacao))
```

Variáveis de Classes e Objetos

```
robo1 = Robot('Automaton 3000')
robo1.diz_oi()
robo2 = Robot('Dr Cyborg')
robo2.diz_oi()
robo1.auto_destruir()
robo3 = Robot('Boomer Prototype')
robo3.diz_oi()
robo2.auto_destruir()
robo3.auto_destruir()
robo1.auto_destruir()
```