Introdução ao Python

Simulação Discreta

Filipe Saraiva

Conteúdo

Conteúdo

Introdução

Para esta disciplina utilizaremos a linguagem **Python**, conforme implementada no interpretador **python3**, para realizar cálculos estatísticos e outros.

Em outro momento também estudaremos o framework de simulação discreta **simpy** durante a disciplina.

Introdução

Por que utilizar Python em Simulação Discreta?

Python é uma linguagem de programação de sintaxe simples e objetiva, que tem diferentes bibliotecas muito utilizadas no campo das ciências em geral, como **numpy**, **scipy**, **pandas**, **matplotlib** e outras. Para a área de simulação discreta, a biblioteca **simpy** é bastante utilizada.

Utilizar Python permitirá ao aluno realização rápida de cálculos utilizados na disciplina além de não depender de software proprietário instalado nas máquinas.

Introdução

Algumas características de Python:

- Multiparadigma (daremos ênfase ao imperativo e orientado a objetos);
- Interpretada;
- Tipagem dinâmica;
- Presença do tipo listas;
- Sintaxe muito próxima de linguagens conhecidas;
- Identação obrigatória;
- Fácil aprendizado.

Conteúdo

Conteúdo

Operadores matemáticos

Alguns operadores matemáticos comuns em Python:

- = $\rightarrow$ atribuição;
- == $\rightarrow$ igual;
- != $\rightarrow$ diferente;
- <, <= $\rightarrow$ menor, menor igual;
- >, >= $\rightarrow$ maior, maior igual.

Conteúdo

Operadores lógicos

Alguns operadores lógicos comuns em Python:

- and $\rightarrow$ condição e;
- or $\rightarrow$ condição ou;
- not $\rightarrow$ condição negativa.

Conteúdo

Definição de Variáveis

Em Python as definições de variáveis seguem a fórmula:

variavel = valor

Por ser de tipagem dinâmica, não há explicitação de tipos na definição de variáveis como existem em outras linguagens (p. ex., em C teríamos *int variavel = valor;*).

Exemplos:

```
x = 2
pi = 3.1416
a, b, c = 1, 2, 3
a, b, c = c, b, a
disciplina = 'Analise de Algoritmos'
lista = [0, 1, 2, 3, 4]
```

Conteúdo

Controle de fluxo

Controle de fluxo são os comandos que alteram o fluxo padrão de execução de um programa. Em Python temos o *if* e suas variantes, *if-else* e *if-elif-else*.

Controle de fluxo – *if*

O *if* padrão testa uma expressão condicional e, caso positivo, executa as instruções dentro do bloco *if*.

Exemplos:

```
if pi == 3.1416:
    print('Valor correto!')
```

Controle de fluxo – *if-else*

O *if-else* testa uma expressão condicional e, caso positivo, executa as instruções dentro do bloco *if*, e caso falso, executa as instruções do bloco *else*.

Exemplos:

```
if pi == 3.1416:
    print('Valor correto!')
else:
    print('Valor incorreto!')
```

Controle de fluxo – *if-elif-else*

Já o *if-elif-else* testa uma expressão condicional para diversos casos, executando apenas para aquele em que a expressão retornará o valor verdadeiro ou, quando atingir o final, executando as instruções do bloco *else*.

Exemplos:

```
if x < 0:
    print('x menor que 0!')
elif x == 0:
    print('x igual a 0!')
else:
    print('x maior que 0!')
```

Conteúdo

Estruturas de Repetição

É comum que linguagens de programação tragam estruturas que permitam executar um conjunto de instruções repetidas vezes. Essas estruturas são normalmente conhecidas como *loops* ou *laços*.

A linguagem Python não é diferente e ela traz as estruturas *for* e *while*.

Estruturas de repetição – *for*

Em Python o laço *for* segue o seguinte padrão:

```
for variavel in sequencia:
    instrucoes
```

Onde *variavel* é a variável que irá iterar (percorrer) sobre a *sequencia*, enquanto *sequencia* é um conjunto de valores que serão atribuídos em ordem à *variavel*.

Existem várias maneiras de definirmos as sequências em Python. Vejamos algumas nos exemplos a seguir.

Estruturas de repetição – *for*

Exemplos:

```
for i in [0, 1, 2, 3, 4]:
    print(i)
```

No exemplo, *i* irá receber os valores da lista na ordem dada, a cada iteração. Portanto, na primeira iteração *i = 0*, na segunda *i = 1*, e assim por diante.

Estruturas de repetição – *for*

Exemplos:

```
lista = [0, 1, 2, 3, 4]
for i in lista:
    print(i)
```

É possível também atribuir os valores para uma variável e colocá-la para ser percorrida pela variável *i* do laço *for*.

Estruturas de repetição – *for*

Exemplos:

```
for i in range(0, 4):
    print(i)
```

Normalmente utiliza-se em Python a função *range*, que cria um intervalo entre valores. Esse intervalo é uma sequência que poderá ser utilizada no *for*, como no exemplo dado onde será criada uma lista *[0, 1, 2, 3]* – o *range* cria a lista até o número antes do último índice.

Estruturas de repetição – *for*

Exemplos:

```
lista = [0, 1, 2, 3, 4]
for i in range(0, len(lista)):
    print(i)
```

Outra função que utilizaremos bastante é a *len*, que retorna o tamanho de uma lista. No caso, combinada com *(*range), teríamos a criação de uma lista que vai de *0* à *4*.

Estruturas de repetição – *while*

Em Python o laço *while* segue o seguinte padrão:

```
while condicao:
instrucoes
```

O *while* fará uma avaliação da *condicao* e, caso verdadeira, executará as instruções que estão dentro do bloco, e ao final avaliará novamente a condição. Essas execuções serão repetidas até que a *condicao* seja falsa, quando o bloco será finalizado.

Estruturas de repetição – *while*

Exemplos:

```
count = 0
while count < 10:
    print(count)
    count = count + 1
```

No exemplo o bloco *while* será executado até que *count* assuma o valor 10, quando então a condição se tornará falsa.

Conteúdo

Blocos

Como já visto nos exemplos anteriores, blocos em Python iniciam com o símbolo do dois pontos *:* e tudo o que vier abaixo disso **e identado** fará parte do bloco.

Exemplo:

```
count = 0
while count < 10:
    print(count)
    count = count + 1
```

O bloco do *while* são todas as instruções após os *:* que estão identados.

Blocos

Exemplo:

```
count = 0
while count < 10:
    print(count)
    count = count + 1
if count == 10:
    print(count)
```

Nesse outro exemplo, a linha condicional *if count == 10:* não está identada dentro do *while*, portanto ela está fora do bloco do *while*.

Funções

Funções em Python são definidas no seguinte padrão:

```
def funcao(parametros):
instrucoes
```

Onde *def* é uma palavra reservada de Python para definições de função, *funcao* é o nome da função, *parametros* é um conjunto de variáveis **sem tipo**, finalizando com *:* que indica o início de um bloco.

Funções

Exemplo:

```
def somar(n1, n2):
    return n1 + n2
```

No exemplo é apresentada uma função *somar*, que recebe dois argumentos e retorna a soma entre eles.

Deve-se ter atenção pois nesse caso não sabemos se os argumentos passados serão sempre números, pois não há tipificação explícita em Python – mas para nossa disciplina, não nos preocuparemos com esse detalhe.

Conteúdo

Conclusões

- Python é interessante para ser utilizado para substituir pseudocódigo em certas disciplinas;
- Utilizaremos a sintaxe de Python durante a disciplina de Análise de Algoritmos;
- Nessa apresentação vimos detalhes importantes da sintaxe que utilizaremos em aulas, como variáveis, estruturas de controle e repetição, operadores, etc.

Introdução ao Python

Simulação Discreta

Filipe Saraiva