Introdução à Linguagem de Programação Python

Tipos de dados, Estruturas Condicionais, Estruturas de Repetição e Funções

Apresentador: Tarik Ponciano

Senac Fecomércio Sesc

Quem é o Apresentador?

Experiências Acadêmicas

Formado em Sistemas e Mídias Digitais pela Universidade Federal do Ceará em 2022.

Artigos publicados nos temas de Computação Contextual e Ensino de Computação.

Monitor de Redes de Computadores, Pesquisador de Acessibilidade e Bolsista de Desenvolvimento PGD.

Experiências Profissionais

Desenvolvedor Júnior Frontend – Fundação Astef – 2021

Instrutor da área de Tecnologia da Informação – Senac – Atual

Bolsista Funcap Graduado no projeto BigDataFor – GREat UFC - Atual

Tecnologias

Python

Html/CSS/Javascript

React.js e React Native

Vue.js e Vue Native

Desenvolvimento Android

REST API e SpringBoot


01
Tipos de Dados
02
Estruturas Condicionais
03
Estruturas de Repetição
04
Funções

Sumário

O que é Python

Python é uma linguagem de programação de alto nível — ou High Level Language —, dinâmica, interpretada, modular, multiplataforma e orientada a objetos — uma forma específica de organizar softwares onde, a grosso modo, os procedimentos estão submetidos às classes, o que possibilita maior controle e estabilidade de códigos para projetos de grandes proporções.

Por ser uma linguagem de sintaxe relativamente simples e de fácil compreensão, ganhou popularidade entre profissionais da indústria tecnológica que não são especificamente programadores, como engenheiros, matemáticos, cientistas de dados, pesquisadores e outros.

Um de seus maiores atrativos é possuir um grande número de bibliotecas, nativas e de terceiros, tornando-a muito difundida e útil em uma grande variedade de setores dentro de desenvolvimento web, e também em áreas como análise de dados, machine learning e IA.

Tecnologias utilizadas na aula

Python 3.11

A mais nova versão estável do Python. Pode ser adquirida no site oficial do Python.

https://www.python.org/downloads/release/python-3110/

Visual Studio Code (IDE)

O editor de código, de código aberto desenvolvido pela Microsoft.

https://code.visualstudio.com/


“
Toda grande caminhada começa com um simples passo
— Buda
Senac
Fecomércio
Sesc

Variáveis e Algoritmos

Variável é o nome utilizado para definir um ou mais valores que são manipulados pelos programas durante a sua operação. O nome “variável” é utilizado por ser um tipo de conteúdo que pode apresentar diferentes valores enquanto o sistema está em execução. Tudo dependerá do comando do usuário e o tipo de operação que é realizado.

Algoritmo é o termo utilizado na área de TI para definir um conjunto básico de regras, variáveis, instruções e condicionantes que definem o funcionamento de um software. Eles funcionam como uma receita de bolo, guiando o computador de acordo com as interações do usuário. Nesse sentido, a estrutura de um algoritmo é moldada da seguinte maneira:

- instruções de entrada, que definem como cada variável deve ser criada de acordo com a ação do usuário;
- instruções operacionais, que definem que ações devem ser feitas de acordo com as ações do usuário;
- instruções de saída, que definem o que deve ser exibido na tela do usuário de acordo com as suas ações.

Tipos de Dados

Python é uma linguagem dinamicamente tipada, o que significa que não é necessário declarar o tipo de variável ou fazer *casting* (mudar o tipo de variável), pois o Interpretador se encarrega disso para nós!
Isso significa também que o tipo da variável poder variar durante a execução do programa.

Os tipos de dados padrão do Python são:

- **Inteiro** (**int**)
- **Ponto Flutuante ou Decimal** (**float**)
- **Tipo Complexo** (**complex**)
- **String** (**str**)
- **Boolean** (**bool**)
- **List** (**list**)
- **Tuple**
- **Dictionary** (**dic**)

Mudar o tipo e erros comuns relacionados ao tipo da variável

Tipos de Dados - Tabela

Example	Data Type
`x = "Hello World"`	str
`x = 20`	int
`x = 20.5`	float
`x = 1j`	complex
`x = ["apple", "banana", "cherry"]`	list
`x = ("apple", "banana", "cherry")`	tuple
`x = range(6)`	range
`x = {"name" : "John", "age" : 36}`	dict
`x = {"apple", "banana", "cherry"}`	set
`x = frozenset({"apple", "banana", "cherry"})`	frozenset
`x = True`	bool
`x = b"Hello"`	bytes
`x = bytearray(5)`	bytearray
`x = memoryview(bytes(5))`	memoryview
`x = None`	NoneType

Senac
Fecomércio
Sesc

Operadores

Operadores são símbolos que dizem ao compilador para realizar manipulações (operações) matemáticas, lógicas e de comparação específicas.

1.Aritméticos

2. Relacionais ou de Comparação

3.Atribuição

4.Lógicos

5. Associação

Operadores - Aritméticos

Operadores aritméticos são usados com valores numéricos para realizar operações matemáticas comuns:

Operator	Name	Example
+	Addition	x + y
-	Subtraction	x - y
*	Multiplication	x * y
/	Division	x / y
%	Modulus	x % y
**	Exponentiation	x ** y
//	Floor division	x // y

Operadores - Comparação

Operadores de comparação são usados para comparar dois valores:

Operator	Name	Example
==	Equal	x == y
!=	Not equal	x != y
>	Greater than	x > y
<	Less than	x < y
>=	Greater than or equal to	x >= y
<=	Less than or equal to	x <= y

Operadores - Atribuição

Os operadores de atribuição são usados para atribuir valores a variáveis:

Operator	Example	Same As
=	x = 5	x = 5
+=	x += 3	x = x + 3
-=	x -= 3	x = x - 3
*=	x *= 3	x = x * 3
/=	x /= 3	x = x / 3
%=	x %= 3	x = x % 3
//=	x //= 3	x = x // 3
**=	x **= 3	x = x ** 3

Operadores - Lógicos

Os operadores lógicos são usados para combinar instruções condicionais:

Operator	Description	Example
and	Returns True if both statements are true	x < 5 and x < 10
or	Returns True if one of the statements is true	x < 5 or x < 4
not	Reverse the result, returns False if the result is true	not(x < 5 and x < 10)

Operadores - Associação

Operadores de associação do Python

Os operadores de associação são usados para testar se uma sequência é apresentada em um objeto:

Operator	Description	Example
in	Returns True if a sequence with the specified value is present in the object	x in y
not in	Returns True if a sequence with the specified value is not present in the object	x not in y

Estruturas Condicionais

Em **Python**, assim como na maioria das linguagens de programação, o programa deve ser capaz de tomar decisões com base em valores e resultados gerados durante sua execução, ou seja, deve ser capaz de decidir se determinada instrução deve ou não ser executada de acordo com uma condição. Para atender a esse tipo de situação, podemos utilizar instruções especiais denominadas **estruturas condicionais**.

- if
- Operadores de comparação
- if-elif-else
- match

Estruturas Condicionais - if

O **IF** é uma estrutura de condição que permite avaliar uma expressão e, de acordo com seu resultado, executar uma determinada ação.

```
media = 7
if media > 6.9:
    print ("Você foi aprovado")
```

No exemplo do código acima, utilizamos apenas o **if** para verificar se a variável media é maior que 6.9. Como esta condição é verdadeira, imprimimos a mensagem na tela “Você foi aprovado”. Caso esta condição fosse falsa, o código seguiria normalmente ignorando, desta forma, a linha 3, o nosso print.

Estruturas Condicionais – if/else

Já o uso o **if/else** fará com que uma das ações sejam executadas, já que se a condição dentro do if não for verdadeira, será executado o código contido no else. O if/else irá testar caso a condição seja verdadeira e executar uma determinada ação ou caso a mesma não seja executar outra.

```
media = 7
if media < 6.9:
    print ("Você foi reprovado")
else:
    print ("Você foi aprovado")
```

O código acima contém dois códigos a serem executados. Caso a media informada seja menor que 6.9, a pessoa será reprovada na disciplina, porém, caso a condição contida no if falhar, o código contido no else será executado. Desta forma, será exibido em tela que o aluno foi aprovado, já que a condição do if é falsa.

Estruturas Condicionais – if/elif/else

O uso de **if/elif/else** serve para quando mais de uma condição precisar ser verificada. Imagine que possuímos duas condições: A primeira, se o aluno possuir uma média menor que 5 e a segunda, se a média for menor que 7 e maior que 5. Vimos anteriormente que utilizamos o if para testar se uma condição é verdadeira, porém, quando precisamos verificar uma segunda condição utilizamos o elif e adicionamos a condição a ser testada.

```
media = 7
if media < 5:
    print ("Você foi reprovado")
elif media > 5 and media < 7:
    print ("Você fará a recuperação")
else:
    print ("Você foi aprovado")
```

No código acima, o primeiro passo é verificar se a média do aluno é menor que 5 e, caso positivo, imprimir a mensagem que o mesmo foi reprovado. Porém, caso essa condição seja falsa, precisamos exibir se ele foi aprovado ou fará a recuperação. Para isso, utilizamos o elif para testar se a média está entre os valores 5 e 7 e, caso positivo, imprimir a mensagem “Você fará a recuperação”. Se a condição contida no if e elif forem falsas, o código contido no else será executado e imprimirá a mensagem “Você foi aprovado”.

Estrutura Bônus:

match

Estruturas de Repetição

Em algumas situações, é comum que uma mesma instrução (ou conjunto delas) precise ser executada várias vezes seguidas. Nesses casos, normalmente utilizamos um loop (ou laço de repetição), que permite executar um bloco de código repetidas vezes, enquanto uma dada condição é atendida.

Em Python, os loops são codificados por meio dos comandos **for e while**. O primeiro nos permite percorrer os itens de uma coleção e, para cada um deles, executar um bloco de código. Já o while, executa um conjunto de instruções várias vezes enquanto uma condição é atendida.

Estruturas de Repetição - for

O laço **for** nos permite percorrer os itens de uma coleção e, para cada um deles, executar o bloco de código declarado no loop. Sua sintaxe é a seguinte:

```
1 | for variavel in lista
2 | comandos
```

Enquanto percorremos a lista de valores, a variável indicada no **for** receberá, a cada iteração, um item da coleção. Assim, podemos executar algum processamento com esse elemento. No código abaixo percorremos a lista **nomes** e imprimimos cada elemento.

```
1 | nomes = ['Pedro', 'João', 'Leticia']
2 | for n in nomes:
3 |     print(n)
```

Estruturas de Repetição - while

O comando **while** faz com que um conjunto de instruções seja executado enquanto uma condição é atendida. Quando o resultado dessa condição passa a ser falso, a execução do loop é interrompida, como mostra o exemplo a seguir:

```
1 contador = 0
2 while (contador < 5):
3         print(contador)
4         contador    = contador + 1
```

Neste código, enquanto a variável contador, inicializada com 0, for menor do que 5, as instruções das **linhas 3** e **4** serão executadas.

Observe que na **linha 4** incrementamos o valor da variável contador, de forma que em algum momento seu valor igualará o número 5. Quando isso for verificado na **linha 2**, o laço será interrompido. Sem esse código, a condição de parada não será atingida, gerando o que é chamado de loop infinito. Evite que isso aconteça, pois leva ao congelamento e finalização da aplicação.

Estruturas de Repetição - while

Se dentro da repetição for executado um break, o loop será encerrado imediatamente.

```
1  x = 0
2  while x < 10:
3      print(x)
4      x += 1
5      if x == 6:
6          print("x é igual a 6")
7          break
```

Você sabia que podemos usar o comando else em estruturas de repetição?

Senac
Fecomércio
Sesc

Funções

Na programação, funções são blocos de código que realizam determinadas tarefas que normalmente precisam ser executadas diversas vezes dentro de uma aplicação. Quando surge essa necessidade, para que várias instruções não precisem ser repetidas, elas são agrupadas em uma função, à qual é dado um nome e que poderá ser chamada/executada em diferentes partes do programa.

A sintaxe de uma função é definida por três partes: nome, parâmetros e corpo, o qual agrupa uma sequência de linhas que representa algum comportamento. No código abaixo, temos um exemplo de **declaração de função em Python**:

```
1 | def hello(meu_nome):
2 |   print('Olá',meu_nome)
```

Funções

Para executar a função, de forma semelhante ao que ocorre em outras linguagens, devemos simplesmente chamar seu nome e passar os parâmetros esperados entre parênteses, conforme o código a seguir.

```
1 >>> hello('Fabio')
2 Olá Fabio
3 >>> meu_nome
4 'Fabio'
```

Funções

Caso seja necessário, também é possível definir funções com nenhum ou vários argumentos, como no código abaixo:

```
1  def hello(meu_nome,idade):
2      print('Olá',meu_nome,'\nSua idade é:',idade)
```

Agora, ao invocar essa função, também é necessário informar o segundo parâmetro, que representa a idade que será impressa após o nome:

```
1  >>> hello('Fabio',28)
2  Olá Fabio
3  Sua idade é: 28
```

Senac
Fecomércio
Sesc

Funções

Assim como podem receber valores de entrada, as funções também podem produzir valores de saída, provenientes de determinadas operações. Nos exemplos anteriores, apenas imprimimos um valor com a função **print**, sem retornar explicitamente um resultado. Abaixo temos uma função que faz o cálculo do salário e retorna o valor a ser pago conforme o número de horas trabalhadas.

```
1  def calcular_pagamento(qtd_horas, valor_hora):
2    horas = float(qtd_horas)
3    taxa = float(valor_hora)
4    if horas <= 40:
5      salario=horas*taxa
6    else:
7      h_excd = horas - 40
8      salario = 40*taxa+(h_excd*(1.5*taxa))
9    return salario
```

Escopo – Variáveis Locais e Variáveis Globais

Local

Uma variável é chamada **local** se ela é **criada ou alterada dentro de uma função**.

Nesse caso, ela existe somente dentro daquela função, e após o término da execução da mesma a variável deixa de existir.

Variáveis parâmetros também são variáveis locais.

Global

Uma variável é chamada **global** se ela for **criada fora de qualquer função.**

Essa variável pode ser visível por todas as funções.

Qualquer função pode alterá-la.

Escopo

O escopo de uma variável determina de quais partes do código ela pode ser acessada, ou seja, de quais partes do código a variável é visível.

A regra de escopo em Python é bem simples:

- As variáveis **globais** são **visíveis por todas as funções.**
- As variáveis **locais** são **visíveis apenas na função onde foram criadas.**

Reveja os conceitos

Tipos de Dados

Primitivos (Int, Float, Complex, Str)

Coleções (List, Tuple, Dictionary)

Estruturas de Repetição

Uso do comando for

Uso do comando While

Operadores

Aritméticos, Atribuição, Comparação, Lógicos, Associação

Funções

Blocos de código com um nome que podem ser executados sempre que chamados. Podem precisar de parâmetros ou não

Estruturas Condicionais

Uso de if-elif-else
A estrutura “match: case”

Escopo

Uso de variáveis globais e locais

Vamos juntar tudo em um programa só!

Senac
Fecomércio
Sesc

Obrigado!

Os materiais desta aula estarão disponíveis em:
https://github.com/TarikPonciano/Bora-La-Senac-Python

Referências

1. https://www.devmedia.com.br/
2. https://www.w3schools.com/python/
3. https://www.treinaweb.com.br/

Senac
Fecomércio
Sesc