



STUDY GUIDE

# Fatec

## Zona Leste



# python

2020

## Study Guide

{ O intuito deste Material é fornecer suporte as aulas de Programação em Python, que ocorrerão todas as quartas-feiras (turma da noite) e segundas-feiras (turma da tarde) das 18h00 às 19h00, destinadas ao apoio da disciplina de Algoritmos e Lógica de programação }

## Ambiente de Desenvolvimento

Repl.it // IDE online utilizada durante as aulas.

{ Outras opções de IDE

Recomendadas:

PyCharm

Sublime Text 2 }

## Professor dirigente

Ricardo Satoshi

//Content created by

Karen Gabriella R.

Dos Santos

## Informações

{ As aulas seguem a ordem dos exercícios do Lote. }

{ O símbolo ‘ # ’ será utilizado nessa apostila para

solicitar atenção a informação. As setas ‘ >> ’

indicam linha de código e entrada/saída de dados. }

Fontes de pesquisa:

Book - Introdução a Programação com Python

Book - Python Básico

Site - DevMedia e eXcript

.....

# .summary

# Study Guide

## Aula 1

- . Variáveis, expressões e comandos
- . Função print
- . Função input
- . Operadores Aritméticos e Lógicos
- . import



## Aula 2

- . Estrutura de Decisão

## Aula 3

- . Estrutura de Repetição

## Aula 4

- . Introdução a Listas//
- . Loop for

## Aula 5

- . Funções

.....

# | Aula 1

## { Variáveis

O gerenciamento de memória no Python é feito de forma automática pelo interpretador da linguagem, o que possibilita que os programadores possam se concentrar mais na resolução do problema, assim eliminando fatores como: declaração de variáveis e sintaxe complexa.

As variáveis são criadas e destruídas ao longo da execução do programa, sua tipagem é dinâmica, isso significa que sua classe é dada através do comando de atribuição; o interpretador observa o comportamento do conteúdo atribuído a variável e assim define o tipo da mesma.

A atribuição é feita através do operador

igual: =

Exemplo:

(comando de atribuição)

↑

Variável = “What’s Up people?”

↓

↓

Referência de Memória

Objeto de classe string (texto)

- # A atribuição é feita somente da direita para a esquerda.
- # A nomeação da variável precisa começar com pelo menos uma letra.
- # Símbolos permitidos na nomeação: acentuação, underline e números.
- # Os espaços não são permitidos.

## { Tipos de Variáveis

O Python trabalha com vários tipos de dados, mas por enquanto vamos manipular inteiros, decimais, textos e variáveis do tipo lógico (verdadeiro ou falso).

Inteiros: `int`

Decimais: `float` (ponto flutuante)

Textos: `str` (strings)

# A utilização de texto deve ser feita entre aspas, simples ou duplas.

Podemos verificar a classe de uma variável através da função: `type()`

## { Função `print()`

A função '`print()`' executa o comando de exibição do conteúdo armazenado nos parênteses, ou seja, realiza uma saída de dados.

// Exemplo:

```
>> print("Bye World!")
>> Bye World!
```



```
main.py  saved
1
2 print("Bye World!")
3
```

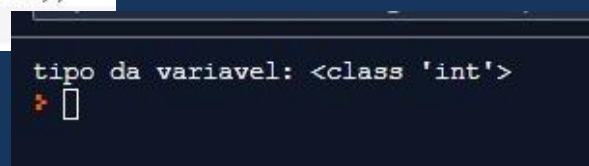
```
https://SomberRealis
Bye World!
```

#Aconcatenação (união) entre variáveis e strings na função `print` é feita através de virgulas.

Exemplo utilizando a função `type`:



```
1
2 variavel = 3
3 print("tipo da variavel:", type(variavel))
4
```



```
tipo da variavel: <class 'int'>
```

## { Função input()

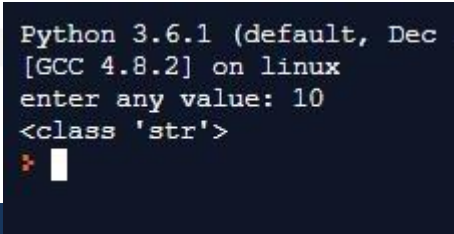
Chamamos de entrada de dados o momento em que seu programa recebe parâmetros (valores) através de um dispositivo de entrada, como por o exemplo: o teclado do computador. Parte relativa ao usuário.

A função input é um comando de entrada de dados; ela recebe um parâmetro, a mensagem a ser exibida para o usuário, e retorna o objeto inserido.

Exemplo:

```
>> Variável = input ( "digite qualquer valor: ")  
>> digite qualquer valor: 10  
>> print(type(Variável))  
>> str
```

```
1 Variável = input("enter any value: ")  
2 print(type(Variável))  
3  
4
```



Como podemos observar no exemplo acima, a variável recebeu o valor inteiro(10) e foi identificada como str através da função type, isso acontece porque a função input sempre retorna valores do tipo string, independente do valor digitado. Devemos converter os tipos retornados de acordo com o esperado pelo programa.

Exemplo:

```
Variável = int(input("digite qualquer valor: "))  
           ↑  
           (conversão para int)
```

```
Variável = float(input("digite qualquer valor: "))  
           ↑  
           (conversão para float)
```

## { Operadores Aritméticos

Sinal	Operação	Exemplo
+	Adição	2 + 3 = 5
-	Subtração	10 - 5 = 5
*	Multiplicação	3 * 4 = 12
/	Divisão	4 / 2 = 2
**	Potenciação	3 ** 3 = 27
%	Resto da divisão	6 % 2 = 0

## { Operadores Lógicos

(*True* = Verdadeiro, *False* = falso)

Sinal	Operação	Exemplo
<b>==</b> (igual a)	Verificação de igualdade entre duas variáveis	3 == 2 False 2 == 2 True
<b>!=</b> (Diferente de)	Verificação de diferença entre duas variáveis	6 != 3 True 5 != 5 False
<b>&gt;</b> (Maior que )	Indica se o valor anterior ao sinal é maior que o posterior	16 > 7 True 4 > 4 False
<b>&lt;</b> (Menor que)	Indica se o valor anterior ao sinal é menor que o posterior	2 < 3 True 8 < 4 False
<b>&gt;=</b> (Maior ou igual)	Indica se o valor anterior ao sinal é maior ou igual ao posterior	4 >= 4 True 3 >= 4 False
<b>&lt;=</b> (Menor ou igual)	Indica se o valor anterior ao sinal é menor ou igual ao posterior	2 <= 2 True 2 <= 3 True 2 <= 1 False

Operador	and (e)	or (ou)	not (não)
True/False	será verdadeiro quando ambas as condições forem verdadeiras.	Será verdadeiro bastando uma condição verdadeira.	Inverte o valor de uma condição.
Exemplo	2 == 2 and 1 == 1 True 3 != 0 and 3 != 3 False	2 != 2 or 1 != 1 False 3 != 0 or 3 != 3 True	Not True False Not false True



## { Import

Para importar um módulo utilizamos o ‘ `import` ’

Exemplo:

```
>> import math
```

```
{ O módulo math fornece  
  acesso às funções  
  matemáticas definidas  
  pelo padrão C. }
```

O código acima importará todos os módulos de `math`, para importar apenas uma parte utilizamos o `from`.

Exemplo:

```
>> from math import sqrt      { sqrt calcula a raiz  
>> print(sqrt(4))            quadrada de um número }  
>> 2
```

Observe que ao utilizar ‘ `from package import item` ’ o `item` pode ser um subpacote, submódulo, classe, função ou variável.

O comando `import` primeiro testa se o `item` está definido no pacote, senão assume que é um módulo e tenta carregá-lo. Se falhar em encontrar o módulo o `ImportError` é lançado.

## | Aula 2

### { Estrutura de Decisão

Para poder escrever programas úteis checar condições é fundamental para determinar o comportamento de uma aplicação. O comando 'if' inicia a estrutura de decisão (determinada por uma condição) seguida de dois pontos ':', que assume caráter verdadeiro ou falso. Se as condições se aplicam o bloco de código indentado é executado, se não, ele encerra e parte para o próximo comando.

++exemplos:

```
2
3  var = int(input("Insira um valor: ")) #entrada de dados
4  if var > 0 :                        #conndição (Se var maior que 0)
5      print("Valor positvo") #bloco de comando indentado
6
7
```

```
Python 3.6.1 (default, Dec
[GCC 4.8.2] on linux
Insira um valor: -2
❏ # Nada acontece.❏
```

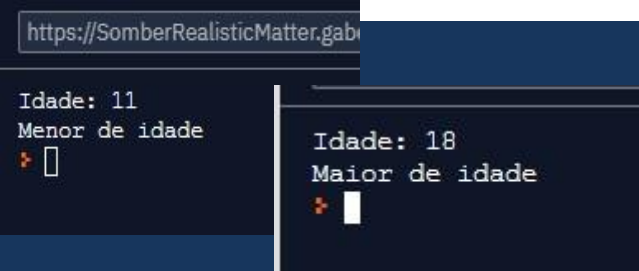
```
Python 3.6.1 (default, Dec
[GCC 4.8.2] on linux
Insira um valor: 5
Valor positvo
❏
```

# O Python é uma das poucas linguagens que utiliza o deslocamento do texto à direita (recuo) para marcar o início e o fim de um bloco de comandos. Outras linguagens contam com as palavras especiais como: BEGIN, END ou as famosas chaves ({ e }) em C e Java.

Um segundo formato da instrução if é a execução alternativa, na qual existem duas possibilidades e a condição determina qual delas será executada. O comando alternativo obedece a instrução 'else:'. Se a primeira condição não for atendida o bloco de comando indentado dentro do else será executado.

## ++exemplos

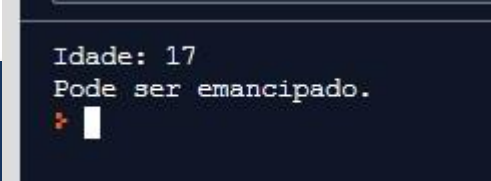
```
main.py  saving...
1
2  idade = int(input("Idade: ")) #entrada de dados.
3  if idade >= 18:                #condição "se"
4      print("Maior de idade")
5  else:                          #instrução "se não"
6      print("Menor de idade")
7
```



Podem existir mais de duas possibilidades e serão necessários mais de dois comandos condicionais, para isso usamos uma condicional encadeada.

## ++exemplos

```
idade = int(input("Idade: ")) #entrada de dados.
if idade >= 18:                #condição "se"
    print("Maior de idade")
elif idade >= 16 and idade < 18: #condição "se não, se"
    print("Pode ser emancipado.")
else:                          #instrução "se não"
    print("Menor de idade")
```



# **elif** é uma abreviação de “else if” (“senão se”).

Não existe um limite para o número de instruções elif, mas se existir uma instrução else ela tem que vir por último. Cada condição é checada na ordem. Se a primeira é falsa, a próxima é checada e assim por diante, se uma delas é verdadeira, o bloco correspondente é executado.

## | Aula 3

### {Estrutura de Repetição

Primeiro, vamos começar definindo o que é iteração: iteração é o processo de repetição onde um bloco de instrução é executado enquanto uma condição for verdadeira, podemos chamar a iteração de laço condicional ou Looping ou mesmo ciclo em português.

Toda linguagem de programação possui no mínimo uma forma de iteração, as estruturas de repetição.

Agora, imagine que pretenda exibir os quadrados dos números 2 até 4. Com o conteúdo estudado até o momento uma solução possível seria:



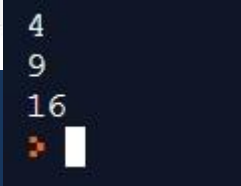
```
main.py saved
1 var = 2 ** 2
2 var2 = 3 ** 2
3 var3 = 4 ** 2
4 print(var, var2, var3)
```

Essa solução resolveria o problema, contudo, se ao invés de 4, fosse necessário exibir os quadrados até 1000 essa solução deixaria de ser simples, assim, a iteração aparece para nos auxiliar a resolver esse tipo de problema.

Uma das estruturas de repetição do Python é o **While**, que executa um bloco de comandos enquanto uma condição for verdadeira, onde condição é uma expressão lógica, e bloco representa as linhas de programa a repetir enquanto o resultado da condição for verdadeiro.

Exemplo de solução utilizando o `while`:

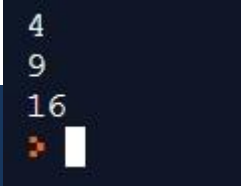
```
i.py saved
var = 2
while var <= 4: #Enquanto var menor ou igual a 4/
    num = var ** 2 #Bloco de comando
    var = var + 1
    print(num)
    # Esse bloco irá se repetir em ordem
    # até que a condição seja satisfeita.
```



Embora muito útil, a estrutura `while` verifica a sua condição de termino no início do laço de repetição. Dependendo do problema, a habilidade de iniciar o `while` sem uma condicional pode ser interessante. Para isso determinamos que o laço vá se repetir infinitamente através do `while True` e utilizamos a instrução `break` para interromper a execução em qualquer parte do bloco.

Exemplo anterior utilizando o `while True`:

```
var = 2
while True: #Sempre verdadeiro
    num = var ** 2 #Bloco de comando
    print(num)
    if var == 4: #condição para terminar
        break
    var = var + 1
```



## | Aula 4

### {Listas

Lista é um conjunto sequencial de variáveis, onde cada variável é identificada através de um índice. A primeira posição da Lista tem índice 0.

Em Python uma lista é declarada da seguinte forma:

```
Nome_da_Lista = [valor1, valor2, valor3..]
```

Uma lista pode ter valores de qualquer tipo, incluindo outras listas.

#### ++exemplos

```
List = ['A',[6,6],8,'B'] #Lista com elementos aleatórios  
print(List[1]) #Exibir em Lista o elemento da posição 1
```

```
[6, 6]
```

Para alterar um elemento da lista, basta fazer uma atribuição de valor através do índice. O valor existente será substituído pelo novo valor.

#### ++exemplos

```
List = ['A',[6,6],8,'B'] #Lista com elementos aleatórios  
List[1] = 7 #Mudança de elemento na posição 1  
print(List)
```

```
['A', 7, 8, 'B']
```

Aprendemos a adicionar itens a uma lista mas, e se fosse necessário produzir uma lista com os números de 1 até 100?

```
L = [1,2,3,4,5..??]
```

Em Python existe a função embutida chamada `range()`, com ela é possível produzir uma lista extensa de uma maneira bem simples:

```
print(list(range(100))) #Mostrar list(função que transforma o range no tipo Lista), range(100)
```

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99]
```

## {Loop for

O `for` é a estrutura de repetição mais utilizada em Python; essa estrutura nos permite percorrer os itens de uma coleção e, para cada um deles, executar o bloco de código declarado no loop.

Sua sintaxe exige, inicialmente, a definição de uma variável e, em seguida, a lista que será iterada.

Exemplo:

```
for variável in lista:  
    bloco de instrução
```

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.

```
nomes = ['Amanda', 'Lucas', 'Jéssica', 'Laura']  
for i in nomes:  
    print(i)
```

```
Amanda  
Lucas  
Jéssica  
Laura  
❖ □
```

O `for` executa um ciclo para cada elemento do objeto que está sendo iterado. Nas vezes em que precisamos que determinada variável seja incrementado ou decrementada a cada ciclo, a forma mais simples, é gerando uma lista com a função `range()`.

Exemplo de números ao quadrado (2 até 4) utilizando o `for in range`:

```
for i in range(2, 5):  
    i = i ** 2  
    print(i)
```

```
4  
9  
16  
❖ □
```



## | Aula 5

### {Funções

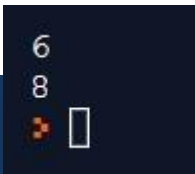
Uma função é um pedaço de código que faz alguma tarefa específica e pode ser chamado de qualquer parte do programa quantas vezes desejarmos.

Sabemos como usar várias funções, como `int`, `float`, `print` e `input`. Neste capítulo, veremos como declarar novas funções e utilizá-las em programas.

Para definir uma nova função, utilizaremos a instrução `def`, seguida pelo nome da função, parênteses para a passagem de parâmetros, dois pontos e o bloco de comandos a ser executado.

Vejamos como declarar uma função de soma que recebe dois números como parâmetros e os imprime na tela:

```
def soma(a,b):  
    print(a+b)  
  
soma(2,4) #chamada função soma  
soma(1,7)
```



```
6  
8  
█
```

No primeiro exemplo, chamamos `soma(2,4)`. Nesse caso, a função será chamada com `a` valendo 2, e `b` valendo 4. Os parâmetros são substituídos na mesma ordem em que foram definidos, ou seja, o primeiro valor como `a` e o segundo como `b`.

Funções são especialmente interessantes para isolar uma tarefa específica em um trecho de programa. Isso permite que a solução de um problema seja reutilizada em outras partes do programa, sem precisar repetir as mesmas linhas. O exemplo anterior utiliza dois parâmetros e imprime sua soma. Essa função não retorna valores como a função `int`. Vamos reescrever essa função de forma que o valor da soma seja retornado:

```
def soma(a,b):  
    return(a+b)  
  
print(soma(2,9))
```

11



Veja que agora utilizamos a instrução `return` para indicar o valor a retornar. Observe também que retiramos o `print` da função. Isso é interessante porque a soma e a impressão da soma de dois números são dois problemas diferentes. Nem sempre vamos somar e imprimir a soma, por isso, vamos deixar a função realizar apenas o cálculo. Se precisarmos imprimir o resultado, podemos utilizar a função `print`, como no exemplo.