Curso Python Avançado

Back to Basics

olist

Linha de Comando
vs
Modo Interativo

Linha de Comando vs Modo Interativo

Executando Script por Linha de Comando

Para executar um script Python contido em um arquivo, basta executar:

- Linux/macOS: `python3 <NOME_DO_ARQUIVO>`
- Windows: `py.exe <NOME_DO_ARQUIVO>`

Arquivos de script Python devem possuir a extensão `.py`.

Também é boa prática sempre incluir um *"Shebang"* na primeira linha de um script (quando se tem a intenção deste script ser executável): `#!/usr/bin/env python3`.

Em sistemas Unix, lembre de dar permissão de execução ao arquivo:
`chmod +x <NOME_DO_ARQUIVO>`.

Linha de Comando vs Modo Interativo

Modo Interativo

Python pode ser executado em modo interativo, neste modo declarações podem ser executados de maneira sucessiva e o resultado da execução pode ser observado imediatamente.

O modo interativo pode ser executado invocando o interpretador por linha de comando sem parâmetros: `python3` / `py.exe`.

Neste modo, `>>>` indica que o interpretador está esperando uma declaração e `...` indica a continuação da declaração, quando este exige múltiplas linhas. Exemplo:

```
>>> print("Hello world!")
Hello world!
>>> the_world_is_flat = True
>>> if the_world_is_flat:
...     print("Be careful not to fall
off!")
...
Be careful not to fall off!
```

Python Basics

Python Basics

Comentários, Declarações e Identação

```
>>> # Isto é um comentário
>>> variavel_1 = # Isto também é um comentário, ao lado de uma
atribuição
>>>               # ... que pode ocupar múltiplas linhas
>>> variavel_2 = "# Não é comentário, está dentro de aspas"
```

Python Basics

Comentários, Declarações e Identação

Blocos de código em linguagens como C++, Java e outras utilizam `{` e `}` para delimitar seu início e fim.

Python utiliza identação.

Um bloco de código (corpo de uma função, loop, etc) começa com identação e termina na primeira linha sem identação.

A quantidade da identação não importa, contanto que seja consistente no bloco.

Em geral 4 espaços em branco são utilizados, *tabulação* deve ser evitada.

Exemplo:

```
>>> for i in range(1, 11):
...     print(i)
...     if i == 5:
...         break
```

Python Basics

Números

Expressões podem ser utilizadas diretamente. Operadores `+`, `-`, `*` e `/` funcionam como na maioria das linguagens e parenteses `()` podem ser utilizados para agrupamento.

```
>>> 2 + 2
4
>>> 50 - 5*6
20
>>> (50 - 5*6) / 4
5.0
>>> 8 / 5  # divisão sempre retorna um número de ponto flutuante
1.6
```

Números inteiros são do tipo `int` e números com parte fracionária são `float`.

Tipos `int` possuem precisão ilimitada, `float` é ... subjetivo ...

Python também possui suporte a números complexos além de outros tipos.

Python Basics

Números

Divisão inteira pode ser feita utilizando o operador `//` e resto de divisão com `%`.

```
>>> 17 / 3  # divisão retorna float
5.666666666666667
>>>
>>> 17 // 3  # divisão inteira discarta a parte fracionária
5
>>> 17 % 3  # operator % retorna resto de divisão
2
>>> 5 * 3 + 2
17
```

Python Basics

Números

O sinal de igual = atribui o resultado da operação a uma variável.

```
>>> largura = 20
>>> altura = 5 * 9
>>> largura * altura
900
```

Python Basics

Strings

Strings podem ser representadas de várias formas. O modo mais comum é utilizando aspas simples ' ou aspas duplas ", ambas possuem o mesmo efeito.

```
>>> 'spam eggs'
'spam eggs'
>>> 'doesn\'t'  # use \' para escape de `
"doesn't"
>>> "doesn't"  # ...ou use aspas duplas
"doesn't"
>>> '"Yes," they said.'
'"Yes," they said.'
>>> "\"Yes,\" they said."
'"Yes," they said.'
>>> '"Isn\'t," they said.'
'"Isn\'t," they said.'
```

Python Basics

Strings

A função `print()` pode ser usada para uma saída mais natural durante o modo interativo.

```
>>> '"Isn\'t," they said.'
'"Isn\'t," they said.'
>>> print('"Isn\'t," they said.')
"Isn't," they said.
>>> s = 'First line.\nSecond line.'
>>> s
'First line.\nSecond line.'
>>> print(s)
First line.
Second line.
```

Python Basics

Strings

Strings literais podem possuir múltiplas linhas e são definidas utilizando três aspas simples ou duplas: `'''...'''` ou `"""..."""`.

```
>>> print("""\
... Usage: thingy [OPTIONS]
...      -h                        Display this usage
message
...      -H hostname               Hostname to connect to
... """)
```

Python Basics

Strings

Strings podem ser concatenadas utilizando o operador `+` e repetidas utilizando o operador `*`.

```
>>> 3 * 'un' + 'ium'
'unununium'
```

Python Basics

Strings

Strings podem ser indexadas, começando por `0`.

```
>>> word = 'Python'
>>> word[0]
'P'
>>> word[5]
'n'
```

Índices também podem ser números negativos. Indexa a partir do fim da string (direita).

```
>>> word = 'Python'
>>> word[-1]
'n'
>>> word[-2]
'o'
>>> word[-6]
'P'
```

Python Basics

Strings

Strings podem ser fatiadas para obter partes de uma string (substring).

```
>>> word = 'Python'
>>> word[0:2]  # characters from position 0 (included) to 2
(excluded)
'Py'
>>> word[2:5]  # characters from position 2 (included) to 5
(excluded)
'tho'
>>> word[:2] + word[2:]
'Python'
>>> word[:4] + word[4:]
'Python'
```

Python Basics

Strings

Strings são imutáveis. De modo que não é possível alterar uma parte delas utilizando, por exemplo, índices.

```
>>> word[0] = 'J'
>>> word[2:] = 'py'
```

A função `len()` pode ser utilizada para obter o comprimento de uma string.

```
>>> s =
'supercalifragilisticexpialidocious'
>>> len(s)
34
```

Exercícios

Exercícios

Exercício 1

Usando a IDE, crie um arquivo com o nome "exercicio_01.py" com o seguinte conteúdo:

```
cars = 100
space_in_a_car = 4.0
drivers = 30
passengers = 90
cars_not_driven = cars - drivers
cars_driven = drivers
carpool_capacity = cars_driven * space_in_a_car
average_passengers_per_car = passengers / cars_driven

print("There are", cars, "cars available.")
print("There are only", drivers, "drivers available.")
print("There will be", cars_not_driven, "empty cars today.")
print("We can transport", carpool_capacity, "people today.")
print("We have", passengers, "to carpool today.")
print("We need to put about", average_passengers_per_car, "in each car.")
```

Exercícios

Exercício 1

Experimentos:

- Troque o valor da variável `space_in_a_car` de `4.0` para `4`.

Exercícios

Exercício 2

Usando a IDE, crie um arquivo com o nome "exercicio_02.py" com o seguinte conteúdo:

```
types_of_people = 10
x = f"There are {types_of_people} types of people."
binary = "binary"
do_not = "don't"
y = f"Those who know {binary} and those who {do_not}."
print(x)
print(y)
print(f"I said: {x}")
print(f"I also said: '{y}'")
hilarious = False
joke_evaluation = "Isn't that joke so funny?! {}"
print(joke_evaluation.format(hilarious))
```

Exercícios

Exercício 2

Experimentos:

- Passe dois argumentos para .format() em joke_evaluation.format(hilarious).
  Ex: .format(hilarious, binary).
- Passe diretamente uma string como argumento para .format() em joke_evaluation.format(hilarious).
  Ex: .format("Maybe...").

Exercícios

Exercício 3

Usando a IDE, crie um arquivo com o nome "exercicio_03.py" com o seguinte conteúdo:

```
print("How old are you?", end=' ')
age = input()
print("How tall are you?", end=' ')
height = input()
print("How much do you weigh?", end=' ')
weight = input()

print(f"So, you're {age} old, {height} tall and {weight} heavy.")
```

Exercícios

Exercício 3

Experimentos:

- Experimente retirar `end=' '` das chamadas a `print()`.
- Experimente o código `age = input("How old are you? ")`.
- O que acontece caso seja usado o código `print("How old are you?" , input())`?