

# Aula 01 - Introdução

## Parte 1 – Revisão Teórica

Um programa em Python pode ser um único arquivo com a extensão \*.py ou então, uma pasta que contém várias sub-pastas com diversos arquivos contendo código Python \*.py e muitos outros contendo informações relevantes ao programa.

Nesse nosso primeiro programa, vamos imprimir uma mensagem na saída padrão, isto é, no console, para que assim, possamos entender um pouco melhor sobre como podemos escrever dados no prompt de comando.

Grande parte do nosso Curso de Python ocorrerá no Prompt de Comando, ou seja, imprimiremos mensagens na tela e em seguida, iremos ler o que o nosso usuário digitou. Essa é a **melhor maneira para o estudo da programação** e também, é a melhor forma para escrevermos software!

### **IMPRIMINDO NA SAÍDA PADRÃO**

Todas as vezes em que desejarmos imprimir uma mensagem na tela, basta utilizarmos a função `print()`, em seguida, abrimos parêntesis, colocamos duas aspas e então, digitar o texto a ser impresso no Console. Feito isso, basta fecharmos o parêntesis e então, executar o código.

Se você já programa/programou em outra linguagem, é importante lembrar que em Python, não finalizamos as instruções com uso do ponto e vírgula!

Ainda que seja permitido a utilização do caractere de finalização ponto e vírgula, não é obrigatório finalizar as instruções com o mesmo. Até podemos colocá-lo se desejarmos, porém, o normal é utilizarmos somente quando precisamos colocar mais de uma instrução numa mesma linha.

Não é correto dizer que não utilizamos ponto e vírgula em Python, até porque, existem situações onde é necessário a sua utilização, como por exemplo, quando precisamos, por alguma razão, escrevermos várias instruções numa única linha. Nesses casos, somos obrigados a utilizar o ponto e vírgula para dizermos ao interpretador onde inicia e onde finaliza cada instrução.

### **A função `print()`: Saída padrão de dados**

A instrução `print` em Python é utilizada todas as vezes em que precisarmos escrever alguma informação na saída padrão, também chamado de Console, linha de comando e etc. A função `print()` pode receber vários parâmetros, como por exemplo, o encoding do texto que será impresso ou então, um nome de arquivo para o qual desejamos escrever dentro.

Inicialmente, vamos utilizar a instrução `print()` para escrever mensagens na saída padrão, isto é, no Console. No trecho de código a seguir, temos um exemplo do uso da função `print()`.

`print()` é uma função nativa do Python. Basta colocar algo dentro dos parênteses que o Python se encarrega de fazer a magia de escrever na tela :)

Em programação, é muito comum utilizar a palavra imprimir (ou `print`, em inglês) como sinônimo de mostrar algo na tela.

```
print("parametro")
```

Usar a letra P maiúscula ao invés de minúscula:

```
>>> Print("Hello, World!")
Traceback (most recent call last):
...
NameError: name 'Print' is not defined
```

Esquecer de abrir e fechar aspas no texto que é passado para a função print():

```
>>> print(Hello, World!)
Traceback (most recent call last):
...
SyntaxError: invalid syntax
Esquecer de abrir ou fechar aspas:
```

```
>>> print("Hello, World!)
Traceback (most recent call last):
...
SyntaxError: EOL while scanning string literal
Começar com aspas simples e terminar com aspas duplas ou vice-versa:
```

```
>>> print('Hello, World!')
Traceback (most recent call last):
...
SyntaxError: EOL while scanning string literal
```

Usar espaço ou tabulação (tab) antes do print():

```
>>> print('Hello, World!')
Traceback (most recent call last):
...
IndentationError: unexpected indent
>>> print('Hello, World!')
Traceback (most recent call last):
...
IndentationError: unexpected indent
```

E como faz para imprimir um texto em várias linhas? Bom, para isso precisamos lembrar de um caractere especial, a quebra de linha: n. Esse n é um caractere especial que significa aqui acaba a linha, o que vier depois deve ficar na linha de baixo.

Por exemplo:

```
>>> print('Olha esse textão sobre aspas simples e duplas.\nIsso aqui é aspas duplas: .→"\nIsso aqui é aspas simples: \'')
```

Olha esse textão sobre aspas simples e duplas.

Isso aqui é aspas duplas: "

Isso aqui é aspas simples: '

Escrevemos o nome da instrução, neste caso, print(). Em seguida, abrimos um parêntesis, abrimos aspas e então, escrevemos a mensagem que deverá ser impressa na tela. Assim, tudo que estiver entre aspas será reconhecido como texto e será impresso no Console. Feito isso, fechamos aspas, fechamos parêntesis e então, podemos executar o nosso programa.

**IMPORTANTE:** tome cuidado para que em todas as vezes que abrires parêntesis fecha-lo ou então, para todas as vezes em que abrires aspas, fechá-las! Do contrário, será levantado uma mensagem de erro pelo interpretador assim que tentares executar o código.

O programa que desenvolvemos possui a base que todo script em Python terá. Logo, ainda que o exemplo tenha sido simples, os princípios aprendidos são muito importantes para o restante do nosso estudo.

```
print("texto 1")
print("texto 2")
```

## COMENTÁRIOS

O uso de comentários é uma prática normal na programação. O objetivo é poder adicionar descrições em partes específicas do código, seja para documentá-lo, seja para adicionar uma descrição, ou mesmo, para marcar que uma determinada linha, ou um conjunto de linhas, não devem ser executados.

Para adicionarmos comentários, utilizamos uma notação especial, a fim de informar ao interpretador que, determinado trecho de caracteres, possui a finalidade de descrever ou documentar algo e assim, o compilador ou interpretador, não deve interpretar os caracteres contidos na notação que demarca trechos de comentários.

## A NOTAÇÃO DEFINIDA PELO PYTHON

Em Python, temos 3 notações utilizadas para definir blocos de comentários. A primeira e mais simples, é para definirmos que uma única linha deve ser ignorada, e as 2 outras, para delimitarmos um conjunto de linhas.

### NOTAÇÃO INLINE

Utilizamos o caractere cerquilha # para demarcarmos que tudo que estiver **a frente** desse caractere, deve ser ignorado pelo interpretador do Python, por exemplo:

```
#Comentario de uma linha, tudo que estiver a frente da cerquilha será ignorado  
print("Aprendendo como utilizar comentários.");  
print("segunda linha");  
print("terceira linha");#aqui é um comentário que não será interpretado
```

exemplo10.py

### NOTAÇÃO MULTILINES (VÁRIAS LINHAS)

A linguagem Python também disponibiliza uma notação para demarcarmos que um conjunto de linhas deve ser ignorado pelo interpretador. Essa notação se dá pelo uso de 3 aspas simples, ou então, 3 aspas duplas. Nestes casos, tudo que estiver delimitado pelas aspas não será interpretado, por exemplo:

```
print("Ola. Python");  
"""  
toda informações contida entre 3 aspas SIMPLES  
é considerada como caracteres que devem ser ignorados.  
"""  
print("Meu 1ro programa em Python");
```

exemplo11.py

No código acima, definimos 2 blocos de códigos que serão ignorados pelo interpretador do Python ou então, podemos dizer que, serão interpretados como se fossem comentários e que dispensam qualquer avaliação por parte do interpretador.

Mas, e se eu precisar usar aspas dentro do texto a ser mostrado na tela? Bem, Caso queira imprimir aspas duplas, envolva tudo com aspas simples e use aspas duplas na parte desejada:

```
>>> print('Python é legal! Mas não o "legal" como dizem pra outras coisas')
```

Python é legal! Mas não o "legal" como dizem pra outras coisas. Caso deseje imprimir aspas simples, faça o contrário (envolva com aspas duplas e use aspas simples onde necessário):

```
>>> print("Python é legal! Mas não o 'legal' como dizem pra outras coisas")
```

Python é legal! Mas não o 'legal' como dizem pra outras coisas

```
print("Bom dia"); print("turma"); print("Python é bem legal! Mas não o 'legal' como dizem pra outras coisas"); print(""); print('O mais "legal" é aprender uma linguagem poderosa e de futuro');
```

exemplo12.py

## PARA QUE UTILIZAMOS COMENTÁRIOS?

A utilização de comentários dá-se tanto para inserirmos umas informações que não podemos esquecer, ou então, algo que desejamos que outros programadores saibam no momento em que estiverem lendo determinadas linhas de código.

A primeira utilização dos comentários é fazer o que o próprio nome sugere: comentar os códigos informando ao interpretador que essas linhas não devem ser executadas.

## DOCUMENTAÇÃO DE PROGRAMAS

Utilizamos a marcação de comentários para documentarmos programas, classes, funções, variáveis, constantes, ou seja, lá o que for. Quando criamos uma biblioteca e essa será utilizada por terceiros, é importante que o código possua uma documentação para que os outros programadores entendam a nossa biblioteca e, saibam como invocar as funções corretamente, ou mesmo, para entenderem o que elas se propõem.

Para a documentação de código, há um conjunto de parâmetros a serem definidos dentro do bloco de comentário a fim de instruir o programa que irá tornar nossos comentários em documentação propriamente dita.

Quando documentamos partes internas do nosso programa, temos que, ao mesmo tempo em que documentamos o que estamos fazendo, também geramos documentação que será posteriormente extraída nosso código e será então, gerada a documentação propriamente dita. Dessa forma, não temos o problema de ter num diretório onde há arquivos que documentam classes e funções, e noutro diretório, as classes e funções documentadas.

Assim, tudo estará junto num mesmo arquivo, feito num mesmo editor! Essa forma de trabalhar, faz com que a documentação sempre esteja atualizada e acelera o desenvolvimento de maneira geral. Até porque, no mesmo momento em que alteramos o código fonte, já podemos alterar a documentação.

É recomendável nos acostumarmos a documentar nosso código. Isso não somente faz de nós melhores programadores como também, ajuda-nos na definir qual a razão que determinado trecho de código está se propondo a realizar.

## INDENTAÇÃO

Indentar é o recuo do texto em relação a sua margem, ou seja, se antes de escrevermos uma instrução, utilizamos 4 espaçamentos da margem esquerda até a instrução propriamente dita, podemos dizer que a indentação utilizada possui 4 espaços.

A palavra indentação, vem do inglês indentation e é normal a sua utilização no código-fonte de um programa, indiferente da linguagem utilizada.

Em Python, a indentação possui função bastante especial, até porque, os blocos de instrução são delimitados pela profundidade da indentação, isto é, os códigos que estiverem rente a margem esquerda, farão parte do primeiro nível hierárquico. Já, os códigos que estiverem a 4 espaços da margem esquerda, estarão no segundo nível hierárquico e aqueles que estiverem a 8 espaços, estarão no terceiro nível e assim por diante.

Todos os blocos são delimitados pela profundidade da indentação e por isso, a sua importância, é vital para um programa em Python. O mau uso, isto é, utilizar 4 espaçamentos enquanto deveríamos estar utilizando 8, acarretará na não execução, ou então, no mal funcionamento em geral.

## BLOCOS

Os **blocos** são uma ou mais instruções que devem ser executadas uma após a outra, de cima para baixo da esquerda para a direita.

Existem vários tipos de blocos, os mais comuns, são os blocos de códigos, isto é, blocos que contenham instruções Python.

Outro tipo comum de bloco são os blocos de comentários, isto é, um conjunto de caracteres, que ocupam uma ou mais linha de código e estão delimitados por uma notação que a linguagem de programação definiu.

## Glossário

### algoritmo (*algorithm*)

Processo geral para solução de uma certa categoria de problema.

### análise sintática (*parse*)

Examinar um programa e analisar sua estrutura sintática.

### bug

Erro em um programa.

### código fonte (*source code*)

Um programa em uma linguagem de alto nível, antes de ter sido compilado.

### código objeto (*object code*)

A saída do compilador, depois que ele traduziu o programa.

### comando `print` (*print statement*)

Instrução que leva o interpretador Python a apresentar um valor na tela.

### compilar (*compile*)

Traduzir todo um programa escrito em uma linguagem de alto nível para uma de baixo nível de uma só vez, em preparação para uma execução posterior.

### depuração (*debugging*)

O processo de encontrar e remover qualquer um dos três tipos de erros de programação.

### erro de semântica ou lógica (*semantic error*)

Erro em um programa, que o leva a fazer algo diferente do que pretendia o programador.

### erro de sintaxe (*syntax error*)

Erro em um programa, que torna impossível a análise sintática (logo, também impossível a interpretação).

### erro em tempo de execução (*runtime error*)

Erro que não ocorre até que o programa seja executado, mas que impede que o programa continue.

### exceção (*exception*)

Um outro nome para um erro em tempo de execução ou erro de *runtime*.

### executável (*executable*)

Um outro nome para código objeto que está pronto para ser executado.

**interpretar (*interpret*)**

Executar um programa escrito em uma linguagem de alto nível, traduzindo-o uma linha de cada vez.

**linguagem de alto nível (*high-level language*)**

Uma linguagem de programação como Python: projetada para ser fácil para os seres humanos utilizarem.

**linguagem de baixo nível (*low-level language*)**

Uma linguagem de programação que é concebida para ser fácil para um computador, tal como a linguagem de máquina ou a linguagem montagem (*assembly language*)

**linguagem formal (*formal language*)**

Qualquer linguagem desenvolvida pelas pessoas para propósitos específicos, tais como, a representação de ideias matemáticas ou programas de computadores; todas as linguagens de programação são linguagens formais.

**linguagem natural (*natural language*)**

Qualquer língua falada pelos seres humanos que tenha evoluído naturalmente.

**portabilidade (*portability*)**

Propriedade que um programa tem de rodar em mais de um tipo de computador.

**programa (*program*)**

Conjunto de instruções que especifica uma computação.

**script**

Um programa guardado em um arquivo (normalmente um que será interpretado).

**semântica (*semantics*)**

O significado de um programa.

**símbolo (*token*)**

Um elemento básico da estrutura sintática de um programa, análogo a uma palavra em uma linguagem natural.

**sintaxe (*syntax*)**

A estrutura de um programa.

**solução de problemas (*problem solving*)**

O processo de formular um problema, encontrar uma solução e expressar esta solução.

## Bibliografia

Python, Escreva seus primeiros programas, Felipe Cruz, Casa do Código.

Curso Introdutório de Python, Grupy-Sanca, 09 de maio de 2019

Tutorial Python, Release 2.4.2. Guido van Rossum, Fred L. Drake, Jr., editor. Tradução: Python Brasil

Curso Python 3. Juracy Filho e Leonardo Leitão. Cod3r.com.br

[https://aprendacompy.readthedocs.io/pt/latest/capitulo\\_01.html](https://aprendacompy.readthedocs.io/pt/latest/capitulo_01.html)

## Parte 2 – Prática (mão na massa)!!!

Para as atividades práticas vamos utilizar o VS Code, que deve estar previamente configurado para executar o python)

Para a criação dos arquivos (com extensão .py) vamos utilizar o editor de texto

Para executar, deve utilizar o interpretador do VS Code, temos 2 opções

- Apertar o Botão direito e selecionar a opção **RUN PYTHON TERMINAL**
- No menu superior, selecionar Debug -> **START WITHOUT DEBUGGING**

Para limpar o console, utilizar o comando clear ou cls

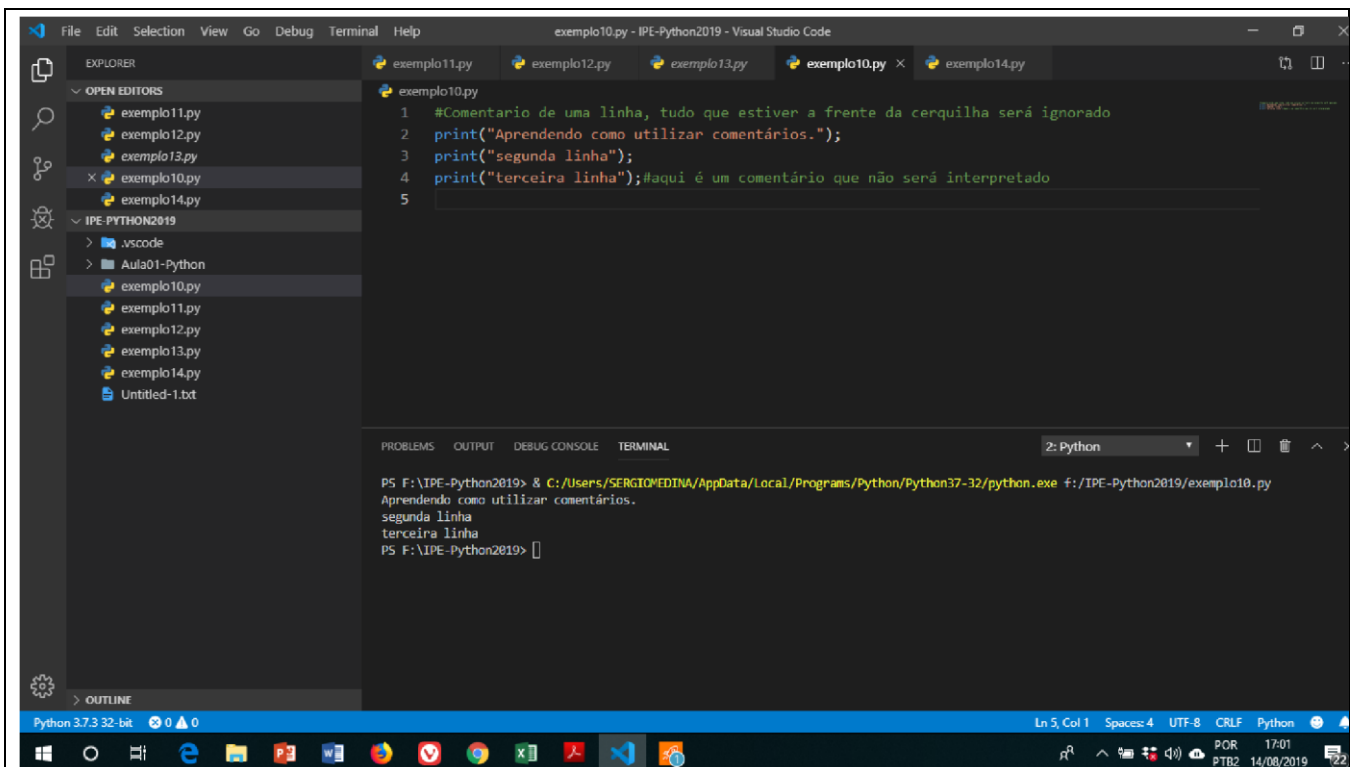
### 2) Criação do Projeto no VS CODE

- Para isto crie uma pasta no seu pendrive com o nome **IPE-Python-2024**.
- Dentro do **IPE-Python-2024**, criar a pasta **Aula01-Python**
- Utilizando o VS CODE acesse a pasta criada (**IPE-Python-2024/ Aula01-Python**)
- Para isso, acione a opção **File > New > File** e crie os seguintes (exemplo10.py, exemplo12.py, exemplo13.py e exemplo14.py), para melhor organização (**BOA PRATICA!!!**)

### Exemplo 10

**PASTA:** IPE-Python-2024/ Aula01-Python

**arquivo:** exemplo10.py



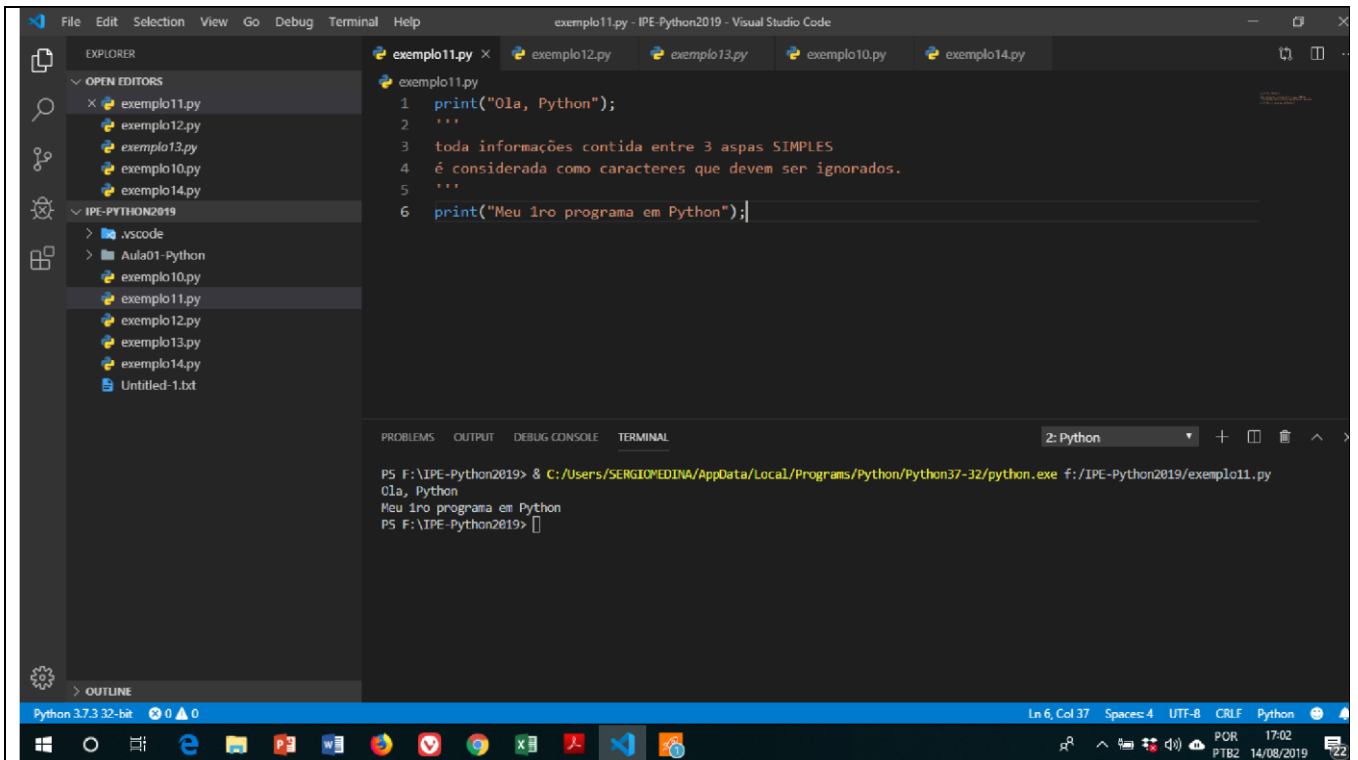
**Figura A:** exemplo11.py



## Exemplo 11

PASTA: IPE-Python-2024/ Aula01-Python

arquivo: exemplo11.py



```
File Edit Selection View Go Debug Terminal Help
exemplo11.py - IPE-Python2019 - Visual Studio Code

EXPLORER
  OPEN EDITORS
    exemplo11.py
    exemplo12.py
    exemplo13.py
    exemplo10.py
    exemplo14.py
  IPE-PYTHON2019
    .vscode
    Aula01-Python
      exemplo10.py
      exemplo11.py
      exemplo12.py
      exemplo13.py
      exemplo14.py
      Untitled-1.txt

OUTLINE

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
2: Python
PS F:\IPE-Python2019> & C:/Users/SERGIOMEDINA/AppData/Local/Programs/Python/Python37-32/python.exe f:/IPE-Python2019/exemplo11.py
Ola, Python
Meu 1ro programa em Python
PS F:\IPE-Python2019> []

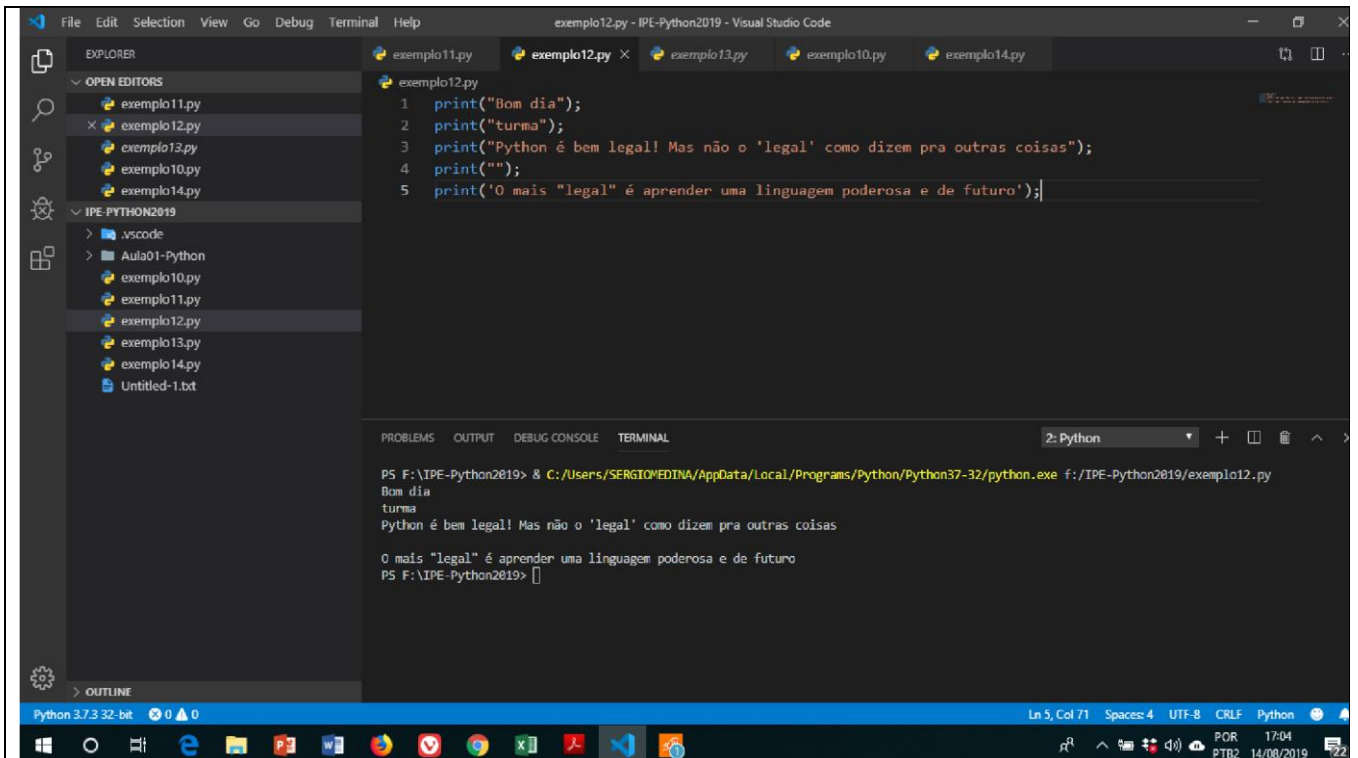
Python 3.7.3 32-bit 0 0 0 Ln 6, Col 37 Spaces: 4 UTF-8 CRLF Python 17:02 14/08/2019
```

Figura B: exemplo11.py

## Exemplo 12

PASTA: IPE-Python-2024/ Aula01-Python

arquivo: exemplo12.py



```
File Edit Selection View Go Debug Terminal Help
exemplo12.py - IPE-Python2019 - Visual Studio Code

EXPLORER
  OPEN EDITORS
    exemplo11.py
    exemplo12.py
    exemplo13.py
    exemplo10.py
    exemplo14.py
  IPE-PYTHON2019
    .vscode
    Aula01-Python
      exemplo10.py
      exemplo11.py
      exemplo12.py
      exemplo13.py
      exemplo14.py
      Untitled-1.txt

OUTLINE

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
2: Python
PS F:\IPE-Python2019> & C:/Users/SERGIOMEDINA/AppData/Local/Programs/Python/Python37-32/python.exe f:/IPE-Python2019/exemplo12.py
Bom dia
turma
Python é bem legal! Mas não o 'legal' como dizem pra outras coisas
O mais "legal" é aprender uma linguagem poderosa e de futuro
PS F:\IPE-Python2019> []

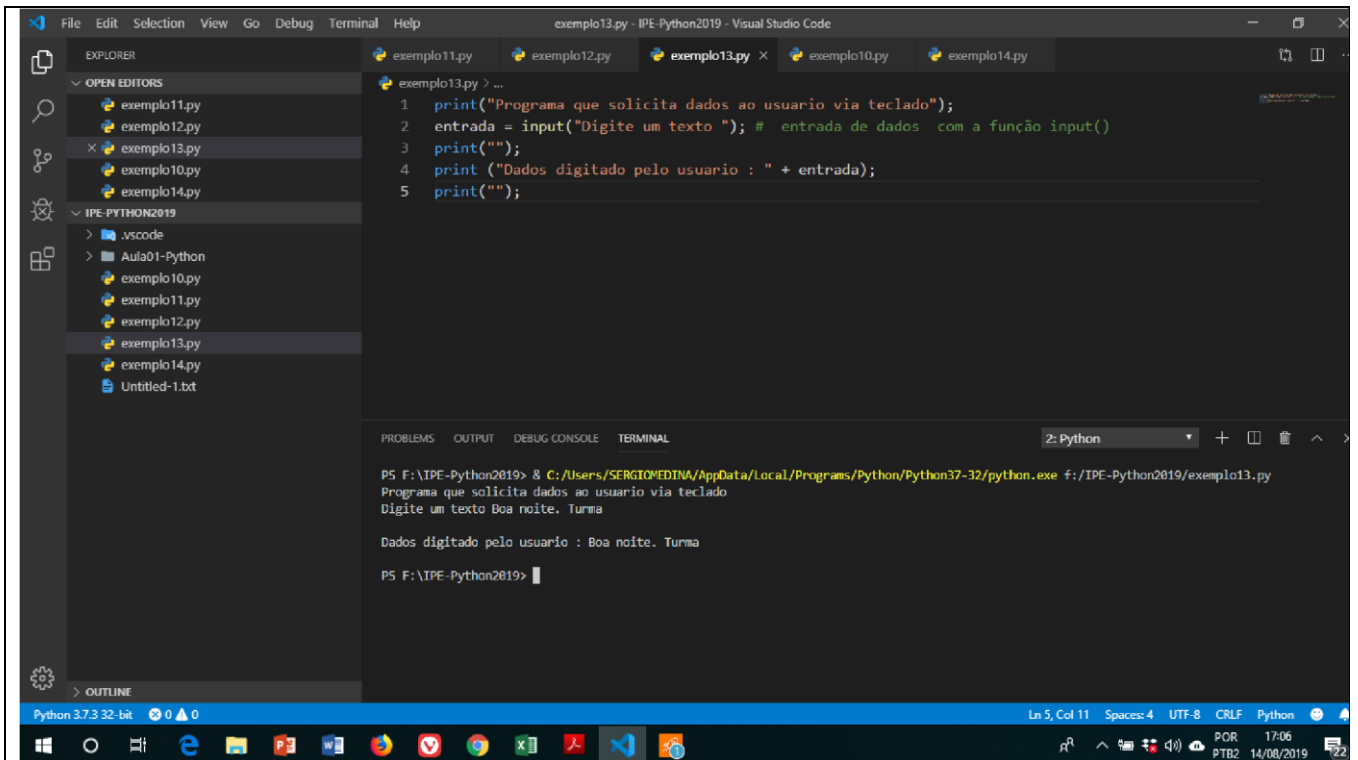
Python 3.7.3 32-bit 0 0 0 Ln 5, Col 71 Spaces: 4 UTF-8 CRLF Python 17:04 14/08/2019
```

Figura C: exemplo12.py



## Exemplo 13

PASTA: IPE-Python-2024/ Aula01-Python  
arquivo: exemplo13.py



```
File Edit Selection View Go Debug Terminal Help
exemplo13.py - IPE-Python2019 - Visual Studio Code

EXPLORER
  OPEN EDITORS
    exemplo11.py
    exemplo12.py
    exemplo13.py
    exemplo10.py
    exemplo14.py
  IPE-PYTHON2019
    .vscode
    Aula01-Python
      exemplo10.py
      exemplo11.py
      exemplo12.py
      exemplo13.py
      exemplo14.py
      Untitled-1.bt

  exemplo13.py > ...
1 print("Programa que solicita dados ao usuario via teclado");
2 entrada = input("Digite um texto "); # entrada de dados com a função input()
3 print("");
4 print ("Dados digitado pelo usuario : " + entrada);
5 print("");

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
2: Python
PS F:\IPE-Python2019> & C:/Users/SERGIO/MEDINA/AppData/Local/Programs/Python/Python37-32/python.exe f:/IPE-Python2019/exemplo13.py
Programa que solicita dados ao usuario via teclado
Digite um texto Boa noite. Turma

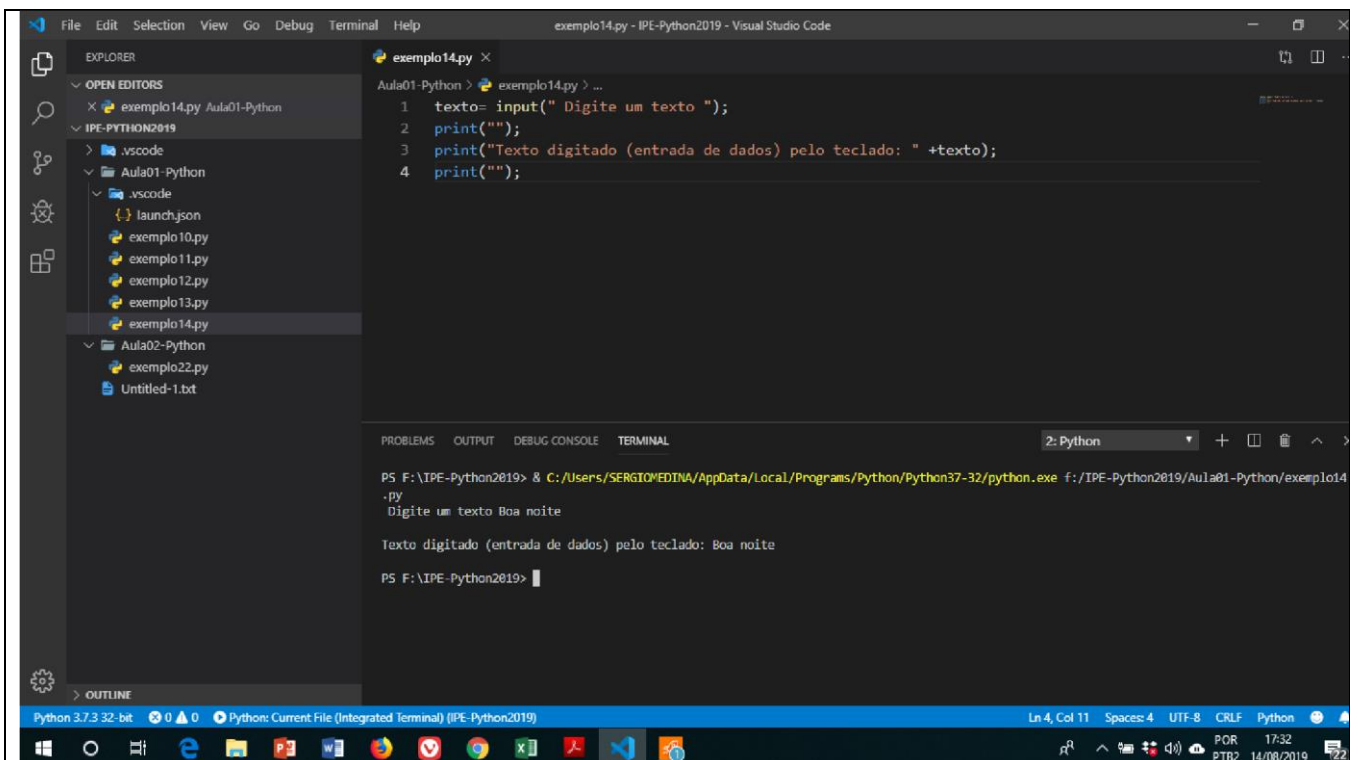
Dados digitado pelo usuario : Boa noite. Turma

PS F:\IPE-Python2019> |
```

Figura D: exemplo13.py

## Exemplo 14

PASTA: IPE-Python-2024/ Aula01-Python  
arquivo: exemplo14.py



```
File Edit Selection View Go Debug Terminal Help
exemplo14.py - IPE-Python2019 - Visual Studio Code

EXPLORER
  OPEN EDITORS
    exemplo14.py
  IPE-PYTHON2019
    .vscode
    Aula01-Python
      exemplo10.py
      exemplo11.py
      exemplo12.py
      exemplo13.py
      exemplo14.py
    Aula02-Python
      exemplo22.py
      Untitled-1.bt

  exemplo14.py > ...
1 texto= input(" Digite um texto ");
2 print("");
3 print("Texto digitado (entrada de dados) pelo teclado: " +texto);
4 print("");

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
2: Python
PS F:\IPE-Python2019> & C:/Users/SERGIO/MEDINA/AppData/Local/Programs/Python/Python37-32/python.exe f:/IPE-Python2019/Aula01-Python/exemplo14.py
Digite um texto Boa noite

Texto digitado (entrada de dados) pelo teclado: Boa noite

PS F:\IPE-Python2019> |
```

Figura E: exemplo14.py