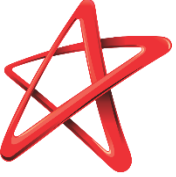


CEUNSP
Centro Universitário
N. Sra. do Patrocínio

CIÊNCIA DA COMPUTAÇÃO
2º SEMESTRE

# Laboratório de Desenvolviemnto de Algoritimos

Prof. Me. Paulo Djalma Martins
paulo.martins@ceunsp.edu.br

CEUNSP
Centro Universitário
N. Sra. do Patrocínio

python


- Python - Histórico
  - 1990
    - Linguagem escrita por Guido Van Rossum
  - 2001
    - Criação da PSF (*Python Software Foundation*)
      https://www.python.org/psf/
  - Características
    - Linguagem de programação interpretada
    - Baseada na linguagem ABC
    - Linguagem de código aberto
    - Linguagem case sensitive

# Trabalhando na plataforma Windows


**IDLE -** Integrated Development and Learning Environment

Desenvolvimento Integrado e ambiente de aprendizagem

O IDLE conhece a sintaxe do Python e ajuda a seguir as suas regras de recuo.

O término com **TAB**

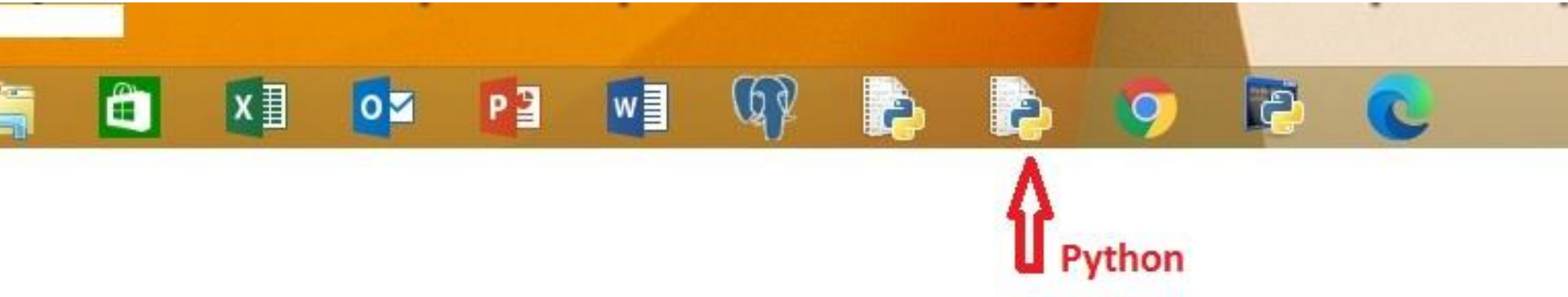
Comece a digitar um código e pressione a tecla **TAB**. O IDLE oferecerá sugestões para ajudar a completar a instrução.

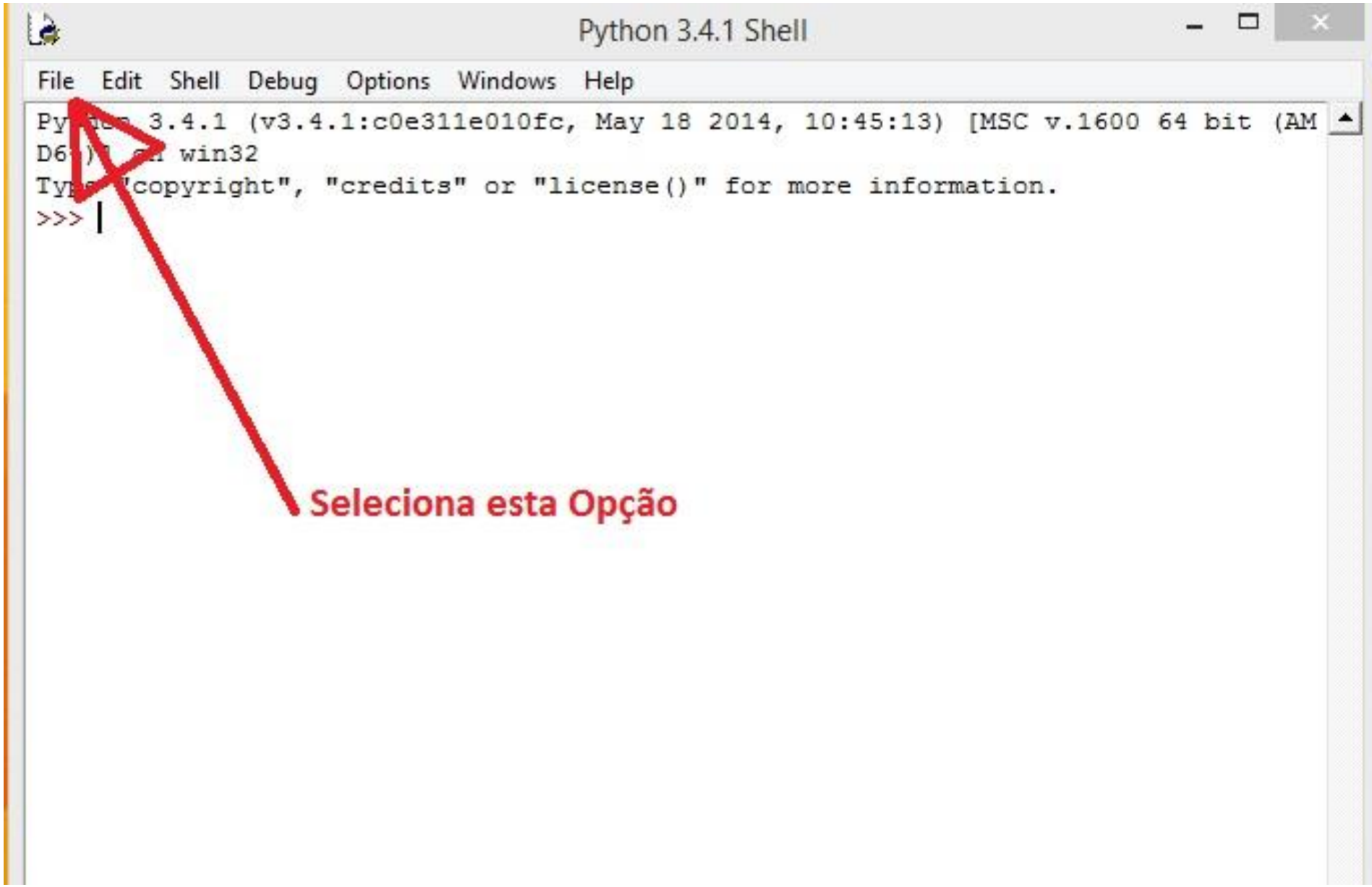
Pressione Alt-P para voltar a instrução anterior

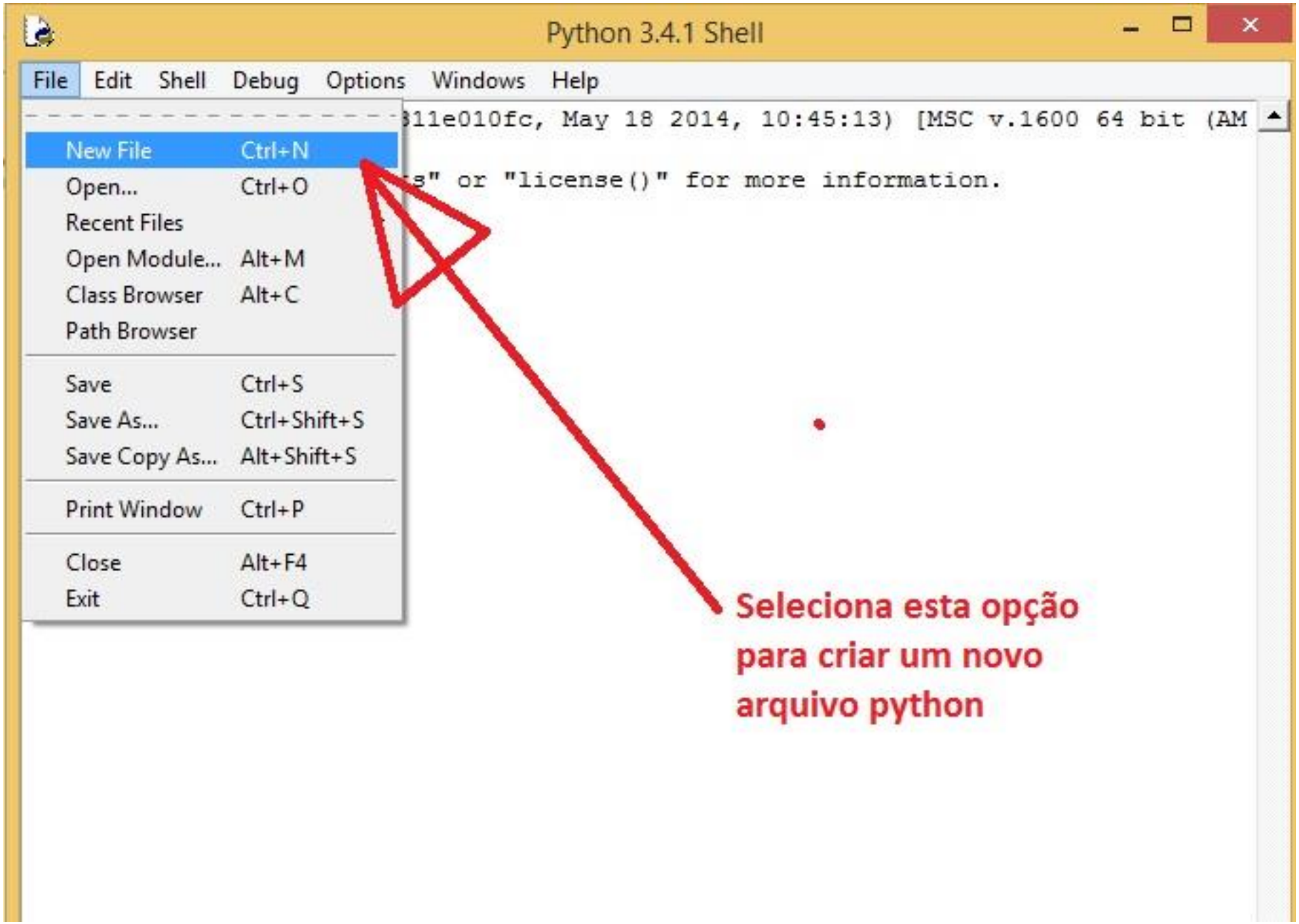
Pressione Alt-N para ir à próxima instrução

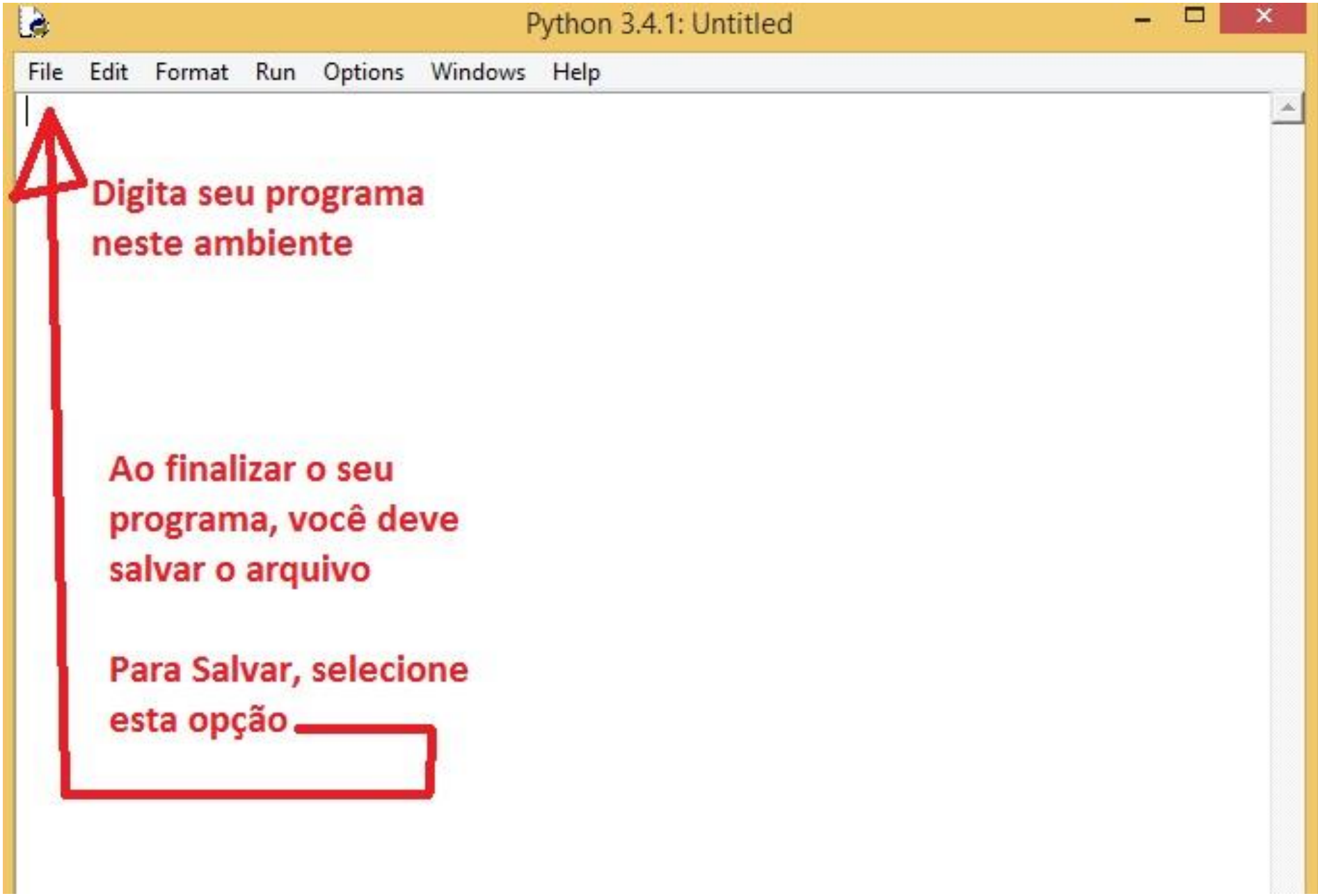
- Ambiente de desenvolvimento
- IDE – *Integrated Development Environment*

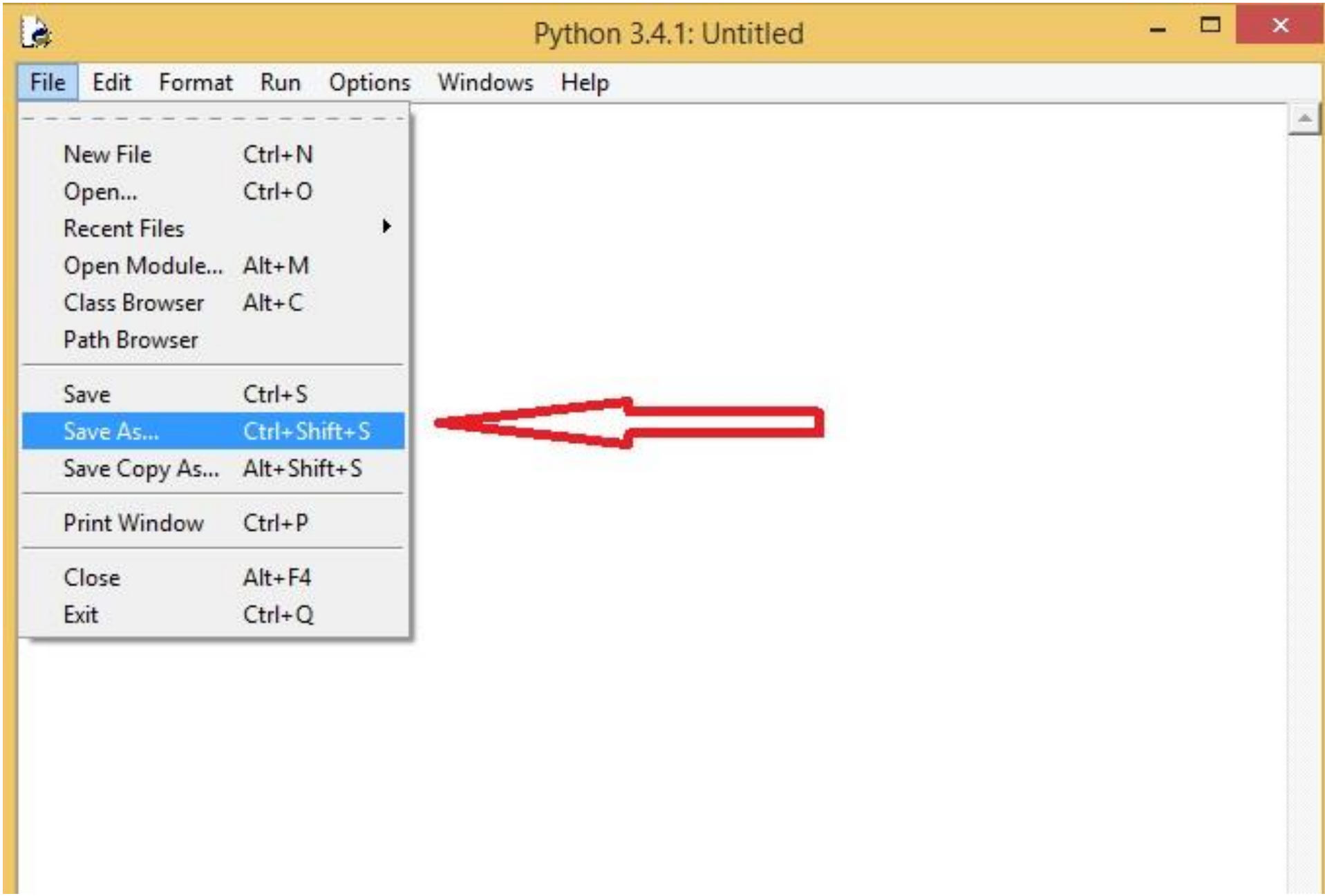
Após a instalação, acessa o IDLE

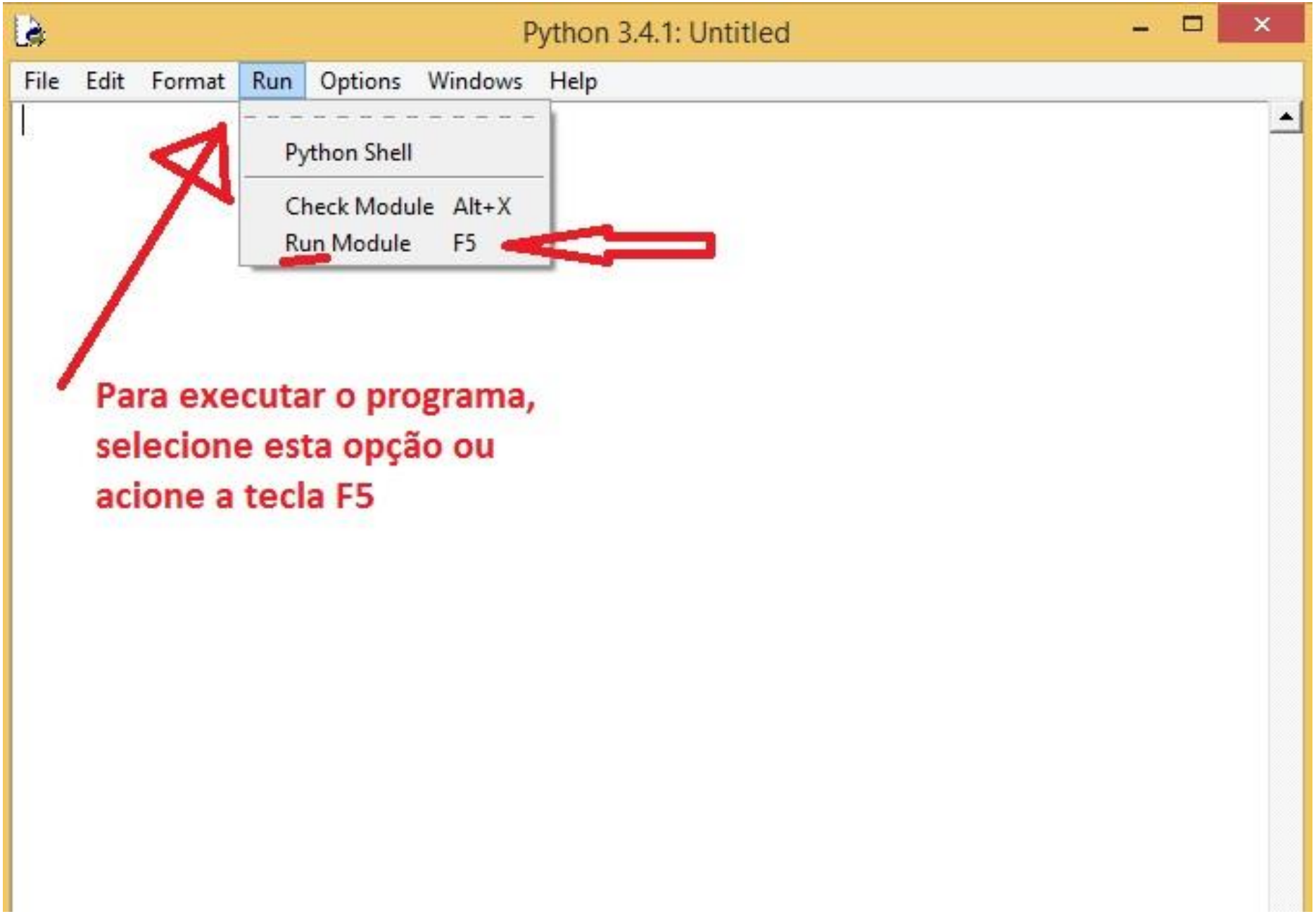



Python 3.4.1 Shell
File Edit Shell Debug Options Windows Help
3.4.1 (v3.4.1:c0e311e010fc, May 18 2014, 10:45:13) [MSC v.1600 64 bit (AM
win32
"copyright", "credits" or "license()" for more information.
>>>
Seleciona esta Opção


Python 3.4.1 Shell
File Edit Shell Debug Options Windows Help
311e010fc, May 18 2014, 10:45:13) [MSC v.1600 64 bit (AM
s" or "license()" for more information.
New File Ctrl+N
Open... Ctrl+O
Recent Files
Open Module... Alt+M
Class Browser Alt+C
Path Browser
Save Ctrl+S
Save As... Ctrl+Shift+S
Save Copy As... Alt+Shift+S
Print Window Ctrl+P
Close Alt+F4
Exit Ctrl+Q
Seleciona esta opção para criar um novo arquivo python


Python 3.4.1: Untitled
File Edit Format Run Options Windows Help
Digita seu programa neste ambiente
Ao finalizar o seu programa, você deve salvar o arquivo
Para Salvar, selecione esta opção


Python 3.4.1: Untitled
File
Edit
Format
Run
Options
Windows
Help
New File Ctrl+N
Open... Ctrl+O
Recent Files
Open Module... Alt+M
Class Browser Alt+C
Path Browser
Save Ctrl+S
Save As... Ctrl+Shift+S
Save Copy As... Alt+Shift+S
Print Window Ctrl+P
Close Alt+F4
Exit Ctrl+Q


Python 3.4.1: Untitled
File Edit Format Run Options Windows Help
Python Shell
Check Module Alt+X
Run Module F5
Para executar o programa,
selecione esta opção ou
acione a tecla F5


- Execução do Python
  - Modo interativo
    - Disponibilização de um prompt para que as instruções sejam digitadas
    - Destinado a processamentos simples
  - Modo direto
    - Execução do interpretador passando como parâmetro de linha de comando o arquivo que contém o código Python
    - Extensão dos arquivos deve ser "**.py**"

- Execução do Python
  - Modo interativo

    Ex: $ python

    >>> mensagem = " exemplo do python"

    >>> print mensagem

  - Modo direto

    Ex: $ python ceunsp.py

| Operadores Aritméticos | |
|---|---|
| **Operador** | **Significado** |
| + | Soma. |
| – | Subtração. |
| * | Multiplicação. |
| / | Divisão (inteira ou ponto flutuante, dependendo dos operandos). |
| ** | Potência. |
| % | Resto da divisão. |
| **Operadores de Comparação** | |
| == | Igualdade. |
| != | Diferença. |
| < | Menor que. |
| > | Maior que. |
| <= | Menor ou igual a. |
| >= | Maior ou igual a. |
| in | Verifica a presença de um elemento numa seqüência. |
| **Operadores de Atribuição** | |
| = | Atribuição. |
| += | Soma com posterior atribuição. |
| –= | Subtração com posterior atribuição. |
| *= | Multiplicação com posterior atribuição. |
| /= | Divisão com posterior atribuição. |
| **Operadores Lógicos** | |
| and | AND lógico. |
| or | OR lógico. |
| xor | XOR lógico. |
| not | NOT lógico. |
| **Operadores sobre Strings** | |
| + | Concatenação. |
| * | Repetição. |
| **Operadores sobre bits** | |
| & | AND. |
| \| | OR. |
| ~ | Inversão em complemento de 2. |
| << | Shift para a esquerda. |
| >> | Shift para a direita. |


- Identação – Controle de Blocos
  - Não importa o tamanho dos espaços, mas sim o alinhamento entre esses espaços

```
Ex 1 :    if numero==10
              numero = numero * 2
              acumulado = acumulado + 1
```

```
Ex 2 :    if numero==10
              numero = numero * 2
          acumulado = acumulado + 1
```

**Obs:** No exemplo 2, apenas a 2ª. Linha esta contido no bloco de controle if


- Identação – Controle de Blocos
  - Instruções de dois blocos

```
Ex 3 :   if numero==10
             numero = numero * 2
             acumulado = acumulado + 1

         if numero==8
             numero = numero+1
             acumulado = 9
```

**Obs:** A Identação facilita a escrita de código mais claro e legível


- Comentário em Python

  - Precede-se o símbolo " # " antes da linha a ser comentada

    Ex : # Programa em Python
    # agosto/2022
    print (" esta é uma linguagem interpretada")


- Variáveis em Python

  - O nome da variável inicia-se com **uma letra do alfabeto**

    Ex :
    **nome, casa01, num → são nomes válidos**

    **89casa, 7iud2, 145op → são nomes inválidos para variáveis em Python**

    **Obs:** Não se pode colocar espaços em nomes de variáveis.
    meu numero, é considerado duas variáveis.


Exemplo:

Variável de escopo específico acessada em outro escopo diferente, sem qualquer vinculação entre eles.

```
Numero = 0

If (numero != 1 ):
        ac=7
        if (ac ==7):
                valor=85
If (ac == 7):
    print (valor)
```


Variáveis

- Tipagem dinâmica:

```
a = 9
a = "ceunsp"
print (a)
```

A variável "a " assume valor numérico e, em seguida, assume um valor textual. A linha print mostrará a ultima entrada na variável "a " .


Variáveis

- função type():

```
var = 9
print ("O tipo da variável \"var\" e:")
print (type (var))

print ("Agora, o tipo da variável \"var\" e: ")
var="teste do Python"
print (type (var))
```

O tipo da variável se adapta automaticamente ao seu contexto de execução


Tipos de dados

- Especificam os valores possíveis que variáveis podem assumir.

Tipo numérico e Booleano:

int → números inteiros

float → números de ponto flutuante

bool → números booleanos

complex → complexos

## Tipos de dados


Exemplo:

```
numero1 = 48
numero2 = 8.0
resultado = numero1/numero2

complexo = 8+2j       #numero complexo
resultado = 0xA4

print (resultado)
```

**Obs:** Os valores inteiros podem ser representados:

- na base decimal
- na base octal ( *nrs. São precedidos po 0* )
- hexadecimal ( *quando os números são precedidos por 0x* )

## Tipos de dados – Variáveis booleanas


Exemplo:

```
condicao = 4 > 6
print (condicao)

condicao = True
print (condicao)
```


Tipos de dados – String

Exemplo:

```
texto1 = "Meu primeiro texto. \n"
texto2 = 'Meu segundo texto'

print (texto1 + texto2)

mensagem = """Veja uma

'mensagem' que pode ser
dividida em mais
de uma "linha"

"""

print (mensagem)
```

**Obs:** " \ " → Carctere de escape


Tipos de dados – String

Exemplo: CARACTERE DE ESCAPE

\n → próximos caracteres na tela deverão iniciar na próxima linha

\" → indica que os caracteres não são strings e devem ser impressos na tela

\t → indica um sinal de tabulação

\\ → indica que o sinal de barra invertida deve ser impresso

# LISTAS

Exemplo:

```
a  = ["Rio de Janeiro", "Natal", "Porto Alegre","Salvador","Manaus"]
print ("A primeiro cidade da lista é: ")

print (a[0] )

print ("Aqui esta a segunda cidade da lista : ")
print (a[1] )

print "Aqui esta a terceira cidade da lista: ")
print (a[2] )

print "Aqui esta a quarta cidade da lista: ")
print (a[3] )

print "Aqui esta a quinta cidade da lista : ")
print (a[4] )
```


Tipo TUPLA

Possuem comportamento semelhante às listas com diferença de que os valores armazenados são imutáveis.

```
Tupla1 = "d", "a", 5 , "python"
Tupla2 = ("hello world", 87, -98)
Tupla3 = ("y",)

print (tupla1 [3])
```


Tipo TUPLA

Sem a vírgula, Python entende ('a') como uma string entre parênteses:

```
>>> t2 = (“a”)
>>> type(t2)
```

```
<type 'string'>
```


## Tipo TUPLA

```
>>> tupla = (“a”, “b”, “c”, “d”, “e”)
>>> tupla[0]

        'a'
```

Selecionando uma “faixa” (*range*) de elementos.

```
>>> tupla[1:3]

('b', 'c')
```


## Tipo TUPLA

Mas se tentarmos modificar um dos elementos de uma tupla, teremos um erro:

```
>>> tupla[0] = "A"
TypeError
```

# Tipo DICIONÁRIO

Possui comportamento semelhante às listas, com a diferença de que os índices dos elementos não precisam ser valores numéricos.

```
pessoa = {"nome":"Daniel", "Idade":30,58:"Python" }

print ("Nome: " + pessoa["nome"])
print ("Idade:" + str(pessoa["Idade"]))
print ("Scripts: " + pessoa[58])
```

Enquanto listas define elementos entre colchetes, o dicionário define entre chaves.

Primeiro é especificado um índice, seguido pelo sinal de dois-pontos e pelo elemento que será armazenado no dicionário.

## Estruturas de Seleção

If

```
entrada = input("Digite um numero de 0 a 100 ")
numero = int(entrada)

if not 0 < numero < 100:
        print ("O numero digitado foi invalido.")

if numero % 2 == 0:
            print ("O numero digitado foi par.")

if numero % 2 != 0:
        print ("O numero digitado foi impar")
```

Cláusula **not** → com esta cláusula, as instruções do bloco de seleção if serão executadas apenas se a condição for avaliada como **falsa**.

% → utilizado para calcular o resto da divisão inteira

Estruturas de Seleção

If

```
entrada = input("Digite um numero de 0 a 100 ")
numero = int(entrada)

if numero < 0 or numero > 100:
        print ("O numero digitado foi invalido.")

if numero % 2 == 0:
        print ("O numero digitado foi par.")

if not numero % 2 == 0:
        print ("O numero digitado foi impar)
```

Cláusula **not** → com esta cláusula, as instruções do bloco de seleção if serão executadas apenas se a condição for avaliada como **falsa**.

% → utilizado para calcular o resto da divisão inteira

Estruturas de Seleção


**If**

```
numero = int(input("Digite um numero de 0 a 100 "))

if numero < 0 or numero > 100:
        print ("O numero digitado foi invalido.")

else:
        print ("O numero digitado foi válido.")
```


## Usando Elif

```
print ("Programa de comparação de números")
numero1=int(input("Digite um número"))
numero2=int(input("Digite outro numero"))

if numero1 > numero2:
      print ("primeiro numero é maior que o segundo numero")
elif numero1 == numero2:
      print ("O primeiro numero é igual ao segundo numero")
else:
      print ("O primeiro numero é menor que o segundo numero")
```


## Estruturas de Repetição

While → executa um conjunto de instruções enquanto uma dada condição for avaliada como verdadeira.

```
entrada = input("Digite a palavra \"sair\"")

while (entrada != "sair"):
        entrada = input("Digite a palavra \"sair\"")

        print ("Fim do programa")
```


Estruturas de Repetição

While

Deixar flexível a digitação do usuário, podendo digitar sair ou SAIR:

```
entrada = input("Digite a palavra \"sair\"")

while (entrada != "SAIR"):
        entrada = input("Digite a palavra \"sair\"")

print ("Fim do programa")
```


Estruturas de Repetição

While

Utilizando um contador para criar um laço com dez repetições.

```
contador = 0

while contador < 10:
        contador += 1
        print (">" * contador )
```

* → Significa repetição para string

Estruturas de Repetição


## For

```
estacoes = [“primavera”,”verao”,”outono”,”inverno”]

for i in estacoes:
        print ( i )
```


Estruturas de Repetição

For

No exemplo abaixo, o laço será repetido dez vezes, com variável var iniciando em 1 e encerrando com valor 10.

```
for var in range(1,10):
    print ("*")
```


## Estruturas de Repetição

Utilizando instrução **continue** e **break**

Para estruturas de laços (for e while), é permitido utilizar instruções especiais.

```
for var in range(1,10):

        if var == 4:
            continue

        if var == 8:
                break

        print ("Numero: " + str(var))
```


Solicitando duas strings e dois número e concatenando:

```
n1 = int(input("Entre com o primeiro número: "))
n2 = int(input("Entre com o segundo número: "))
string1 = input("Entre com a primeira string: ")
string2 = input("Entre com a segunda string: ")

print (string1 + string2)
print (n1 * n2)
```


Métodos de Lista

len() → calcular quantos itens de dados existem na lista

```
forevers = ["Classe", 'Emerson', 'Almeida', 'Alessandro']
print (forevers)

print (len(forevers))

print (forevers[1])
```


Métodos de Lista

append() → adicionar um único item de dados do final da lista

pop() .....→ remover os dados do final da lista

extend() → adicionar uma coleção de dados ao final da lista

remove() → encontra e remove um item especifico da lista

insert() → adiciona um item de dado antes da posição específica

append() → adicionar um único item de dados do final da lista

```
forevers = [“Classe”, ‘Emerson’, ‘Almeida’, ‘Alessandro’]
print (forevers)

forevers.append (“Kyota”)

print (forevers)
```

pop() .....→ remover os dados do final da lista

```
print (forevers)

forevers.pop()

print (forevers)
```


Métodos de Lista

extend() → adicionar uma coleção de dados ao final da lista

```
print (forevers)

forevers.extend (["Alex Almeida", "Felipe Valeta"] )

print (forevers)
```


## Métodos de Lista

remove() → encontra e remove um item especifico da lista

```
print (forevers)

forevers.remove("Emerson")

print (forevers)
```

insert() → adiciona um item de dado antes da posição específica

```
print (forevers)

forevers.insert(0,"Emerson")

print (forevers)
```

# Bibliografia

1) Administração de Redes com Scripts

Costa, Daniel Gouveia

Brasport

2) Python

Barry, Paul

Alta Books, 2012

3) Python3 – Conceitos e Aplicações

Banin, Sergio Luiz

Érica, 2018