



Apostila de Introdução à Programação **Prof. Eduardo Corrêa**

Capítulo IV –Programação Python – Conceitos Elementares

Sumário

IV. Programação Python – Conceitos Elementares.....	2
IV.1 Linguagem Python	2
IV.2 Escrevendo, Salvando e Executando Programas Python	3
IV.3 Variável: o elemento fundamental da programação.....	9
IV.4 Entrada de dados	17
IV.5 Saída de dados	19
IV.6 Problemas resolvidos.....	21
IV.7 Funções matemáticas – módulo ‘math’	25
Exercícios propostos	28



IV. Programação Python – Conceitos Elementares

Na unidade anterior trabalhamos os primeiros conceitos sobre **lógica de programação** e **especificação de algoritmos**. Foram apresentados exemplos que demonstraram como aplicar técnicas que, de maneira simples e natural, nos possibilitam encadear pensamentos e separá-los corretamente em uma sequência de passos com o intuito de solucionar um determinado problema computacional. A partir de agora, você vai aprender a transformar algoritmos (a alma de um programa) em programas reais (a materialização do algoritmo) elaborados em Python. Para iniciar, este capítulo apresenta os conceitos básicos da programação Python: variáveis, tipos, operações de atribuição, operações aritméticas, funções matemáticas e comandos de entrada e saída de dados. Alguns desses assuntos já haviam sido antecipados no Capítulo I, porém serão agora explicados em maior nível de detalhe. As estruturas de seleção e repetição ainda não serão apresentadas na presente unidade, constituindo o tema das Unidades V e VI, respectivamente.

IV.1 Linguagem Python

Python é uma linguagem de programação de **propósito geral**, o que significa que através dela é possível construir qualquer tipo de programa, desde um programa simples que calcula o dobro de um número, até programas complexos um sistema de inteligência artificial. A linguagem foi criada no ano de 1991, tendo como principal filosofia priorizar a construção de programas simples e legíveis (aquilo que os *pythonistas*, ou seja, os programadores Python, chamam de programas “bonitos”) sobre a construção de programas que, ainda que velozes, sejam complicados e pouco legíveis (os ditos programas “feios”). No decorrer dos dez anos seguintes, a linguagem alcançou grande popularidade tanto em ambiente acadêmico como corporativo. Isto motivou o surgimento da *Python Software Foundation* no ano de 2001, uma instituição independente e sem fins lucrativos que tornou-se responsável pelo desenvolvimento de novas versões da linguagem (no momento da elaboração desta edição do livro, a versão mais recente era a 3.10.4).

Nos últimos anos, Python consolidou-se como uma das tecnologias mais difundidas na área de estatística / ciência de dados, apesar de não ter sido originalmente projetada para este fim. Isto se deve principalmente às seguintes características da linguagem:

- Como Python é uma linguagem **interpretada**, os iniciantes podem aprender alguns comandos e começar a fazer coisas legais (ex.: aplicar funções matemáticas e estatísticas sobre conjuntos de dados) quase que imediatamente, sem esbarrar em problemas relacionados à compilação de código. E para tornar a coisa ainda melhor, o interpretador Python pode ser utilizado de forma **interativa**, onde cada comando digitado é imediatamente traduzido e executado. Isto oferece aos programadores uma forma ágil e simples para examinar em tempo real os resultados intermediários obtidos em qualquer passo de um processo de análise de dados.
- Python é uma **linguagem livre** (*open source*). No Web site da Python Software Foundation¹ é possível baixar gratuitamente o arquivo que instala o interpretador Python e a sua biblioteca padrão (a famosa *standard library*). Juntos, estes



Escola Nacional de Ciências Estatísticas

componentes formam o “coração” do ambiente Python, oferecendo um rico conjunto de estruturas de dados (como listas e dicionários) e centenas de módulos voltados para a execução dos mais diversos tipos de tarefas, desde o uso de funções matemáticas e estatísticas até o processamento de bases de dados em diferentes formatos (CSV, JSON, etc.).

- A linguagem Python pode ser facilmente estendida através da incorporação de outros **pacotes**. Atualmente, existem milhares de pacotes disponíveis no repositório central do Python (*Python Package Index* – PyPI). Muitos e muitos deles são voltados para estatística e ciência de dados, tais como, ‘NumPy’ (manipulação de vetores e matrizes), ‘SciPy’ (rotinas numéricas para resolver problemas de integração, equações algébricas e cálculo numérico, entre outras coisas), ‘pandas’ (importação e transformação de bases de dados), ‘Matplotlib’ (geração de gráficos) e ‘scikit-learn’ (algoritmos de mineração de dados e aprendizado de máquina).
- A linguagem Python é uma das mais didáticas de todas as linguagens, talvez a mais fácil de se aprender. Linguagens como C, C# e Java (também muito populares) são **altamente simbólicas**. Nestas linguagens, tem-se por exemplo que os símbolos “{“ e “}” são utilizados para delimitar o início e o fim de um bloco de código subordinado a comandos de seleção ou repetição. A linguagem Python, por outro lado, adota o **espaçamento**, a mesma abordagem que normalmente é utilizada para elaborar algoritmos em pseudocódigo. Desta forma, os principiantes acabam por se sentir menos assustados ao lidar com o Python como uma primeira linguagem.

IV.2 Escrevendo, Salvando e Executando Programas Python

Há várias formas de criar, modificar, salvar, testar e executar programas Python. Neste curso, recomendamos a utilização do **ambiente Thonny** (apresentado na Unidade I, Seção I.3-C e I.3-D), uma vez que ele é mais simples do que os outros ambientes de desenvolvimento e foi criado especificamente para apoiar o ensino de programação. Na segunda parte do nosso curso, apresentaremos algumas alternativas ao Thonny, tais como IDLE, Google Colab, e outras.

Na maioria dos ambientes, incluindo o Thonny, os programas Python podem ser executados em dois modos: **script** (ou **normal**) e **interativo**. No modo script, o programa é primeiro digitado por completo, em seguida salvo e, por fim, executado “de cabo a rabo” (isto é, por inteiro, sem pausas) pelo interpretador Python. De maneira oposta, quando estamos trabalhando no modo interativo, o Python interpreta e executa comando por comando, conforme eles vão sendo digitados. A receita para trabalhar com o Thonny nos modos script e interativo é apresentada nas subseções a seguir.

IV.2.1 Utilizando o Thonny no modo script

- **PASSO 1:** no seu computador pessoal, execute o Thonny efetuando o duplo-clique em seu ícone na área de trabalho (Figura 1). Se você estiver no laboratório, entre na pasta “C:\Thonny” e efetue o duplo-clique no executável “thonny.exe” (Figura 2).



Escola Nacional de Ciências Estatísticas


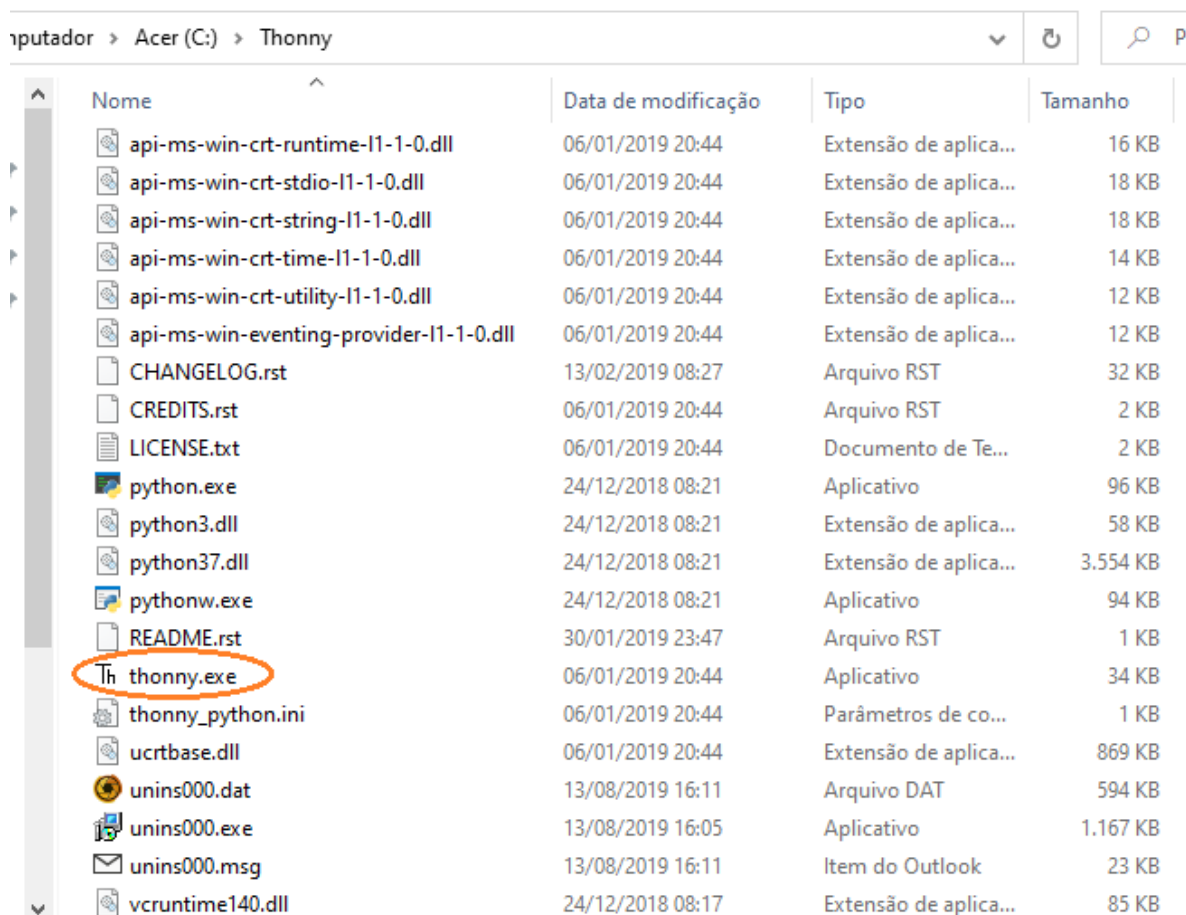
Dependendo da configuração de seu computador, a extensão “.exe” pode não ser exibida (você verá apenas a palavra “thonny” ao lado do ícone ).



Figura 1. Em seu computador pessoa, procure o ícone do Thonny na área de trabalho para executá-lo.




Computador > Acer (C:) > Thonny				
Nome	Data de modificação	Tipo	Tamanho	
api-ms-win-crt-runtime-l1-1-0.dll	06/01/2019 20:44	Extensão de aplica...	16 KB	
api-ms-win-crt-stdio-l1-1-0.dll	06/01/2019 20:44	Extensão de aplica...	18 KB	
api-ms-win-crt-string-l1-1-0.dll	06/01/2019 20:44	Extensão de aplica...	18 KB	
api-ms-win-crt-time-l1-1-0.dll	06/01/2019 20:44	Extensão de aplica...	14 KB	
api-ms-win-crt-utility-l1-1-0.dll	06/01/2019 20:44	Extensão de aplica...	12 KB	
api-ms-win-eventing-provider-l1-1-0.dll	06/01/2019 20:44	Extensão de aplica...	12 KB	
CHANGELOG.rst	13/02/2019 08:27	Arquivo RST	32 KB	
CREDITS.rst	06/01/2019 20:44	Arquivo RST	2 KB	
LICENSE.txt	06/01/2019 20:44	Documento de Te...	2 KB	
python.exe	24/12/2018 08:21	Aplicativo	96 KB	
python3.dll	24/12/2018 08:21	Extensão de aplica...	58 KB	
python37.dll	24/12/2018 08:21	Extensão de aplica...	3.554 KB	
pythonw.exe	24/12/2018 08:21	Aplicativo	94 KB	
README.rst	30/01/2019 23:47	Arquivo RST	1 KB	
 thonny.exe	06/01/2019 20:44	Aplicativo	34 KB	
thonny_python.ini	06/01/2019 20:44	Parâmetros de co...	1 KB	
ucrtbase.dll	06/01/2019 20:44	Extensão de aplica...	869 KB	
unins000.dat	13/08/2019 16:11	Arquivo DAT	594 KB	
unins000.exe	13/08/2019 16:05	Aplicativo	1.167 KB	
unins000.msg	13/08/2019 16:11	Item do Outlook	23 KB	
vcruntime140.dll	24/12/2018 08:17	Extensão de aplica...	85 KB	

Figura 2. No laboratório, execute o arquivo thonny.exe dentro da pasta C:\Thonny

- **PASSO 2:** A tela da Figura 3 será aberta. A parte **superior** é o **editor de código**, onde você escreve seus programas. A área inferior, denominada **shell**, é onde os **resultados**, ou seja a **saída do programa**, serão apresentados.

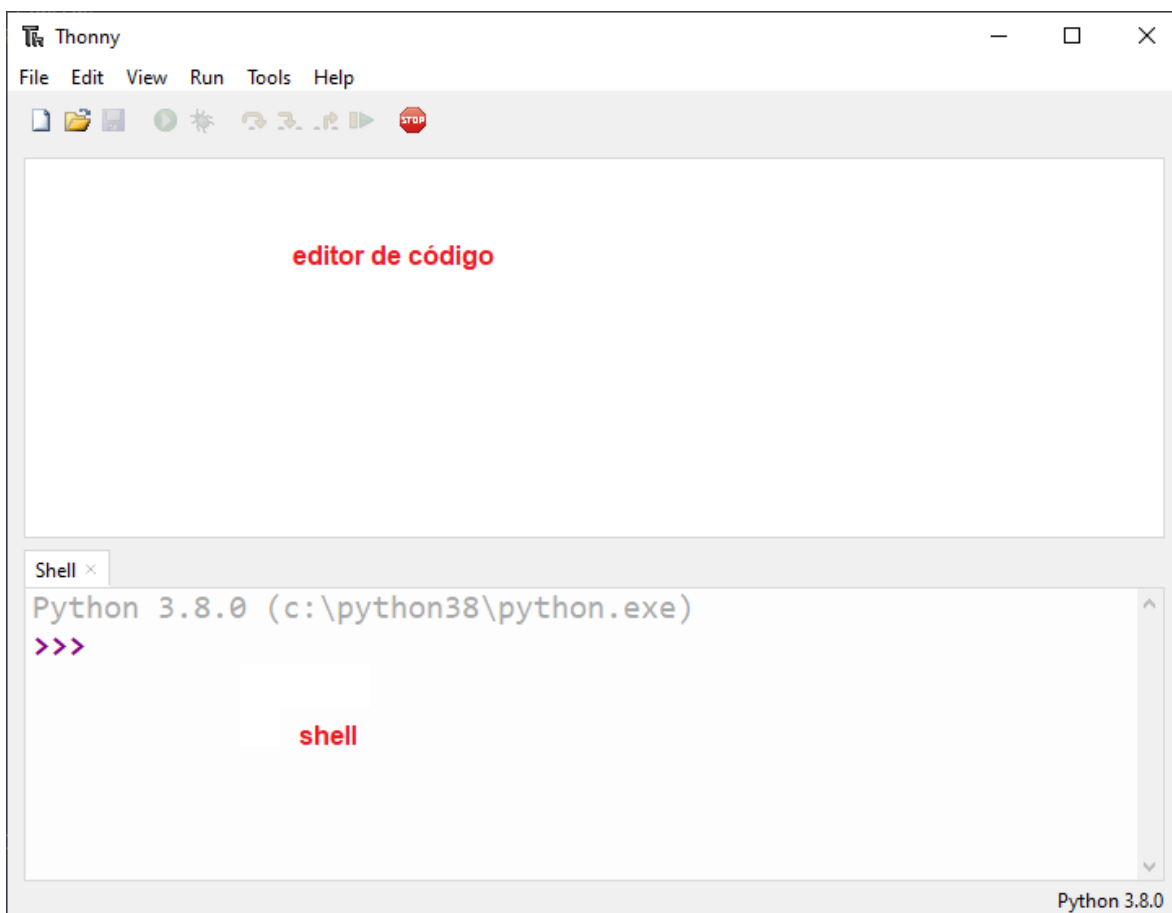


Figura 3. Tela principal do Thonny, dividida em editor de código (parte superior) e shell (parte inferior)

- **PASSO 3:** para criar um novo programa, clique no primeiro ícone no canto superior esquerdo da tela (desenho da folha em branco) ou escolha a opção de menu File / New, como na Figura 4. Com isso, uma nova aba de código chamada 'untitled' será aberta (Figura 5).

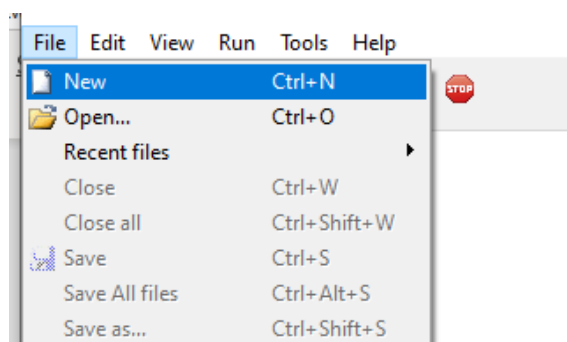


Figura 4. Use File / New para criar um novo programa



Figura 4. Nova aba para digitação de código

- **PASSO 4:** agora basta digitar o seu programa e, quando terminar, clicar no botão salvar (destacado na Figura 5) para que o programa fonte, ou seja **o texto que você digitou no editor de código**, seja salvo como um arquivo em alguma pasta do seu computador.
 - O programa será salvo com a extensão “.py”, isto é, se você escolher o nome “ola_ence”, ele será salvo como “ola_ence.py”.
 - Ao salvar o programa, o nome da aba com o editor de código mudará de “untitled*” para o nome que você escolheu ao salvar o programa.
- **PASSO 5:** a execução do programa pode ser feita de diferentes formas.
 - Você pode clicar no ícone semelhante ao botão play (destacado na Figura 6);
 - Digitar a tecla F5;
 - Ou escolher a opção de menu Run / Run current script.

Seja qual for a sua escolha, o resultado (saída) do programa será exibido na Shell, como mostra a Figura 6. Se você mandar executar um programa ainda não salvo, o Thonny vai exigir com que o mesmo seja salvo antes de executá-lo.



Escola Nacional de Ciências Estatísticas

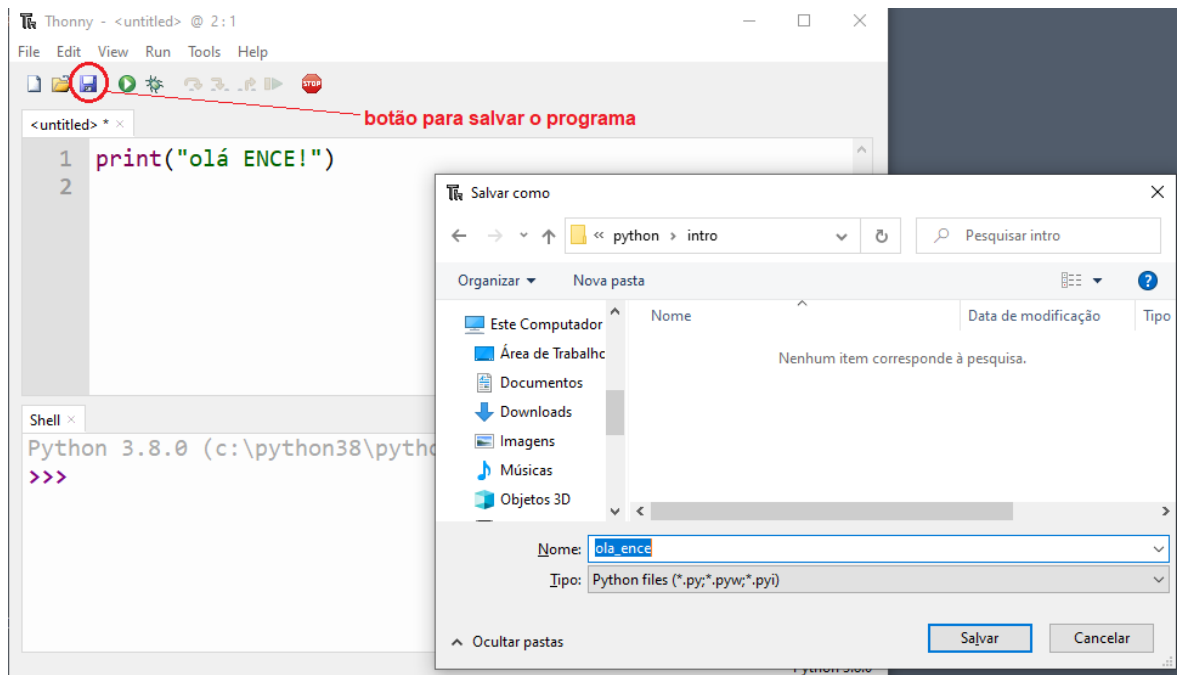


Figura 5. Salvando o programa



Figura 6. Executando o programa e visualizando os resultados.

- **ABRINDO PROGRAMAS:** a qualquer momento você pode abrir programas já salvos no Thonny. Basta escolher a opção de menu File / Open. Cada programa será aberto



Escola Nacional de Ciências Estatísticas

em uma aba diferente, conforme ilustrado na Figura 7, em que há 3 diferentes programas abertos no Thonny. Para fechar um programa, basta fechar a sua aba.

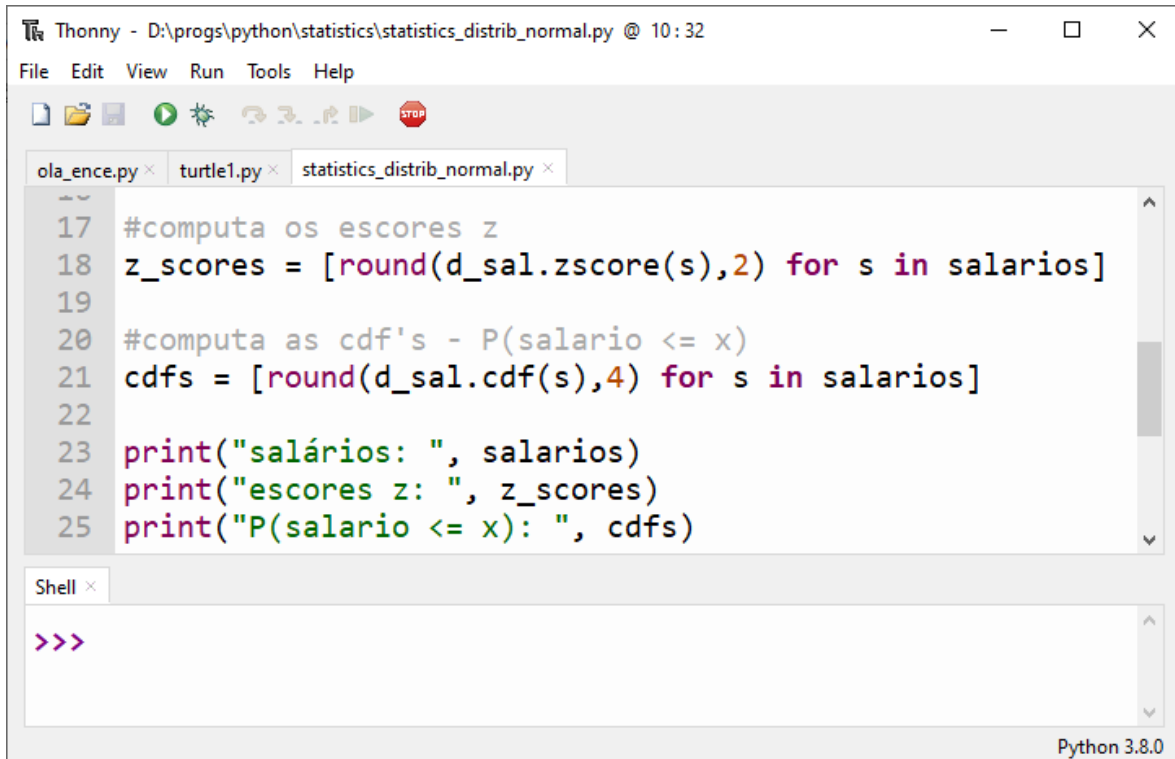


Figura 7. Thonny com vários programas abertos, cada qual em uma aba diferente

IV.2.2 Utilizando o Thonny no modo interativo

- **PASSO 1:** basta executar o Thonny (da mesma forma descrita na subseção anterior) e digitar os comandos **diretamente na Shell**. Cada comando que você digitar será imediatamente traduzido e executado. Veja um exemplo na Figura 8.
 - O primeiro comando digitado na Shell foi `1 + 1`. Após você digitar a tecla ENTER, o Python irá traduzi-lo e executá-lo automaticamente, exibindo o resultado 2 na própria Shell.
 - O segundo comando digitado foi `print("Olá ENCE!")`. Ao digitar ENTER, o Python vai traduzir e executar o comando, exibindo a o resultado na Shell.
 - E assim vai ocorrer para qualquer comando no modo interativo.
- **IMPORTANTE:** no nosso curso, vamos utilizar o Python principalmente no modo script e não no interativo.

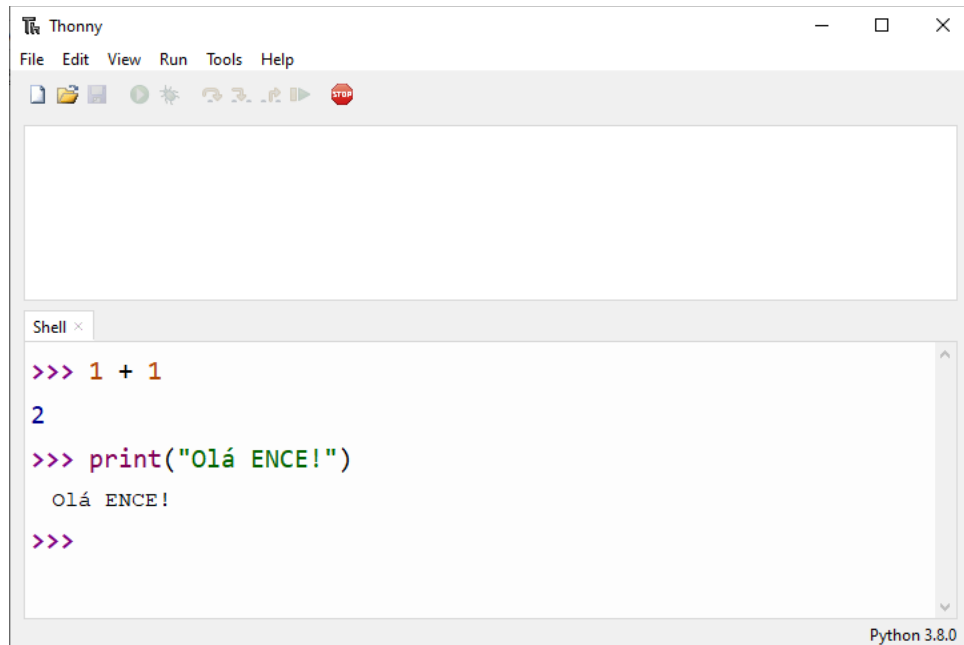


Figura 8. Trabalhando com o Python no modo interativo

Agora que você já sabe como criar e executar os seus programas, chegou a hora de conhecer os conceitos fundamentais sobre a programação em Python, começando pelo conceito de variável e tipos.

IV.3 Variável: o elemento fundamental da programação

O conceito de variável pode ser entendido pelo aluno da seguinte forma: o nome de um **local onde é possível guardar um determinado valor**. Este valor pode ter sido especificado como entrada pelo usuário (via teclado, por exemplo) ou pode ter sido calculado dentro do programa. Para simplificar ainda mais, você pode imaginar a variável como sendo uma “caixinha” que possui um nome e guarda um determinado valor (Figura 9).

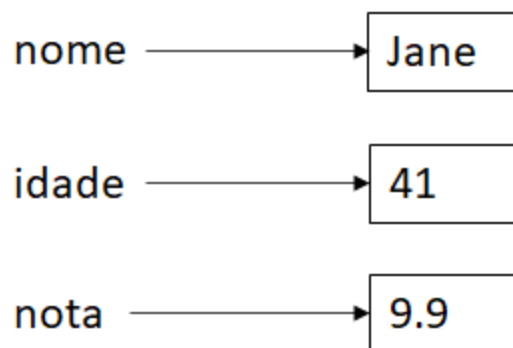


Figura 9. Variáveis servem para guardar informações.



Escola Nacional de Ciências Estatísticas

Na figura acima são apresentadas três variáveis: “nome”, “idade” e “nota”. A variável “nome” armazena o valor ‘Jane’. Em outras palavras, pode-se dizer que o **conteúdo** da variável “nome” é ‘Jane’. A variável “idade” tem conteúdo 41, enquanto “nota” armazena o valor 9.9.

Na realidade uma variável representa um **endereço da memória principal** do computador, onde alguma informação está armazenada (Figura 10). Melhor explicando: a variável é um “apelido” para uma posição de memória. É mais fácil, dentro de um programa, localizar a informação usando este apelido, por exemplo, “nome”, do que utilizando o número de um endereço onde a informação está armazenada como, por exemplo $(\text{FFFFEA00})_{16}$.

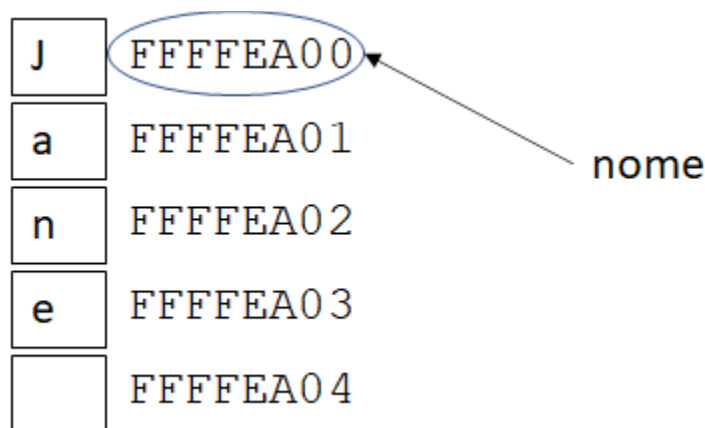


Figura 10. Uma variável aponta para algum endereço da memória principal.

O código a seguir exemplifica o uso de variáveis em programas Python e apresenta os tipos de dado básicos oferecidos pela linguagem. Neste exemplo tanto como nos demais apresentados ao longo da apostila, mostra-se primeiro o código do programa e, logo abaixo, o resultado da execução. O resultado estará sempre listado após o prompt “>>>”.

```
#P001: variáveis e tipos; funções type() e print()

#PARTE 1: declaração de variáveis
nome = 'Jane'
sobrenome = "Austen"
idade = 41
nota = 9.9
aprovado = True

#PARTE 2: imprimindo o conteúdo das variáveis e os tipos das mesmas
print(nome, sobrenome, idade, nota, aprovado)
print(type(nome))
print(type(sobrenome))
print(type(idade))
print(type(nota))
print(type(aprovado))
```



Escola Nacional de Ciências Estatísticas

```
#PARTE 3: mudando o valor e o tipo da variável "nota"
nota = 'A'
print('mudei o valor e o tipo de "nota" para: ', nota, ",", type(nota))
```

```
>>>
Jane Austen 41 9.9 True
<class 'str'>
<class 'str'>
<class 'int'>
<class 'float'>
<class 'bool'>
mudei o valor e o tipo de "nota" para: A , <class 'str'>
```

Para começar, é importante mencionar que o **fluxo de execução** de qualquer programa, ou seja, a ordem em que os comandos são executados pelo interpretador Python, é de cima para baixo, em sequência. O programa foi dividido em três partes. Vamos às explicações sobre cada uma:

- Na PARTE 1, encontram-se as declarações das variáveis “nome”, “sobrenome”, “idade”, “nota” e “aprovado”. Estas variáveis são declaradas, respectivamente, com valores do tipo **str** (abreviação de **string**, isto é, alfanumérica, tanto “nome” quanto “sobrenome”), **int** (valores inteiros), **float** (valores reais) e **bool** (valores lógicos: True ou False – mais detalhes em breve). Estes são os quatro **tipos de dados primitivos** do Python. O nome “primitivo” refere-se ao fato de que estes são os tipos de valores mais simples com os quais a linguagem trabalha.
 - O **operador de atribuição**, representado pelo símbolo “=”, é usado para atribuir um valor (lado direito da instrução) a uma variável (lado esquerdo). Por exemplo: `nota = 9.9`. O valor associado não precisa ser sempre fixo – ou um **literal**, usando o jargão da informática; ele pode também ser outra variável (por exemplo, `a = b`) ou uma expressão (`perimetro_do_quadrado = 4 * lado`).
 - Em Python, a declaração de variáveis pode ser realizada em qualquer linha do programa. O tipo da variável será automaticamente determinado de acordo com o valor que lhe for atribuído, podendo mudar durante a execução do programa caso o valor da variável seja alterado.
 - Os valores de strings podem ser definidos entre aspas simples (`nome = 'Jane'`) ou aspas duplas (`sobrenome = "Austen"`).
- A PARTE 2 cobre o uso de `print()`, a função padrão para saída de dados. Para imprimir diversas variáveis, basta separá-las por vírgula. Observe que valores de qualquer tipo podem ser impressos com esse comando. Ainda nesta parte, utilizamos



Escola Nacional de Ciências Estatísticas

a função `type()`, uma função muito útil que retorna o tipo de dado armazenado em uma variável.

- Na PARTE 3, mostra-se como uma variável pode ter não apenas o seu valor, mas também seu tipo modificado durante o fluxo de execução. Neste trecho de código, o valor de “nota” é mudado de 9.9 (tipo float) para ‘A’ (tipo str).

Comentários em Programas Python

Comentários são textos ou frases definidos com o uso do símbolo `#`. No programa anterior, observe que a primeira linha contém um texto com comentários:

```
#P001: variáveis e tipos; funções type() e print()
```

Os comentários são ignorados pelo interpretador Python na hora em que o programa é processado. Eles servem apenas para documentação, isto é, para serem lidos pelos humanos que estiverem analisando ou alterando o programa.

Na prática, é interessante utilizar comentários para descrever o objetivo de um programa (neste caso, podem ser colocados nas linhas iniciais), para explicar partes mais complicadas, para documentar as variáveis, etc. Comentários são uma excelente ferramenta para deixar o seu programa bem organizado e documentado, mas é importante utilizá-los com bom senso! Você não precisa encher o seu código de comentários e nem comentar coisas óbvias. Por exemplo, o comentário abaixo é desnecessário, pois é óbvio:

```
x = 0 #atribui o valor 0 para a variável x
```

IV.3.1 Nomenclatura de Variáveis

Quando o programador for dar nome para as variáveis em um programa Python, precisará respeitar as seguintes regras:

- Os nomes de variáveis podem conter letras, números e o caractere *underscore* (sublinhado), no entanto não podem ser iniciados por um número.
- Com relação ao underscore, nós até podemos começar nomes com ele, mas apenas em situações específicas – portanto, evite fazer isso por enquanto!

Os seguintes nomes de variáveis são **válidos**:

- peso, altura, x, y, z, km, Milhas, zzz, PI
- salario_anual, tot_renda, z_z_z
- n1, n2, z10, H999

Os nomes a seguir são **inválidos** (não podem ser utilizados como nomes de variáveis):

- 1n, 10_abc, 2altura (começam com número).
- *peso*, E@mail, &1, x\$ (contém caracteres especiais).
- salario anual (possui espaço em branco)



Escola Nacional de Ciências Estatísticas

Há **diferenciação entre letras maiúsculas e minúsculas**, o que significa que se você nomear uma variável como “x”, não poderá referenciá-la como “X”. (mas lembre-se de que a recomendação é usar minúsculo nos nomes de todas as variáveis). A documentação da linguagem Python sugere que todas as variáveis devam ter o nome especificado em minúsculo e que o caractere underscore (_) seja utilizado para separar nomes compostos, padrão este conhecido como snake case. Alguns exemplos:

- idade
- renda_media_anual
- codigo_ocupacao
- produtos_2022

O nome de uma variável também **não pode** ser igual a nenhuma das **palavras reservadas** do Python – uma lista de palavras que possui um papel especial na linguagem. São elas:

```
['False', 'None', 'True', 'and', 'as', 'assert', 'break', 'class',  
'continue', 'def', 'del', 'elif', 'else', 'except', 'finally', 'for',  
'from', 'global', 'if', 'import', 'in', 'is', 'lambda', 'nonlocal', 'not',  
'or', 'pass', 'raise', 'return', 'try', 'while', 'with', 'yield']
```

IV.3.2 Tipo de Dado

Toda variável possui, obrigatoriamente, um **tipo de dado**. O tipo de uma variável determina a **espécie de valor** que a esta pode armazenar e é automaticamente definido sempre que a variável é declarada através de um comando de atribuição. Em Python, há quatro tipos primitivos (ou básicos) considerados mais importantes: **str**, **int**, **float**, e **bool**.

(a) tipo **str**

As variáveis do tipo str – abreviação de string – podem armazenar qualquer informação alfanumérica, isto é, textos de qualquer tamanho, incluindo textos que possuem números. Para definir uma variável do tipo str, basta realizar um comando atribuição colocando o valor (ou literal, que é o lado direito da atribuição) entre aspas simples ou duplas. Abaixo alguns exemplos de comandos de atribuição que definem variáveis do tipo str:

```
letra = 'A'  
endereco = "Rua André Cavalcanti, 106 - Bairro de Fátima"  
empresa = 'ACME LTDA'  
s = '@'
```

(b) tipo **int**

As variáveis do tipo int armazenam números inteiros, negativos ou positivos. Para definir uma variável deste tipo, o processo é similar ao que você faria no papel: basta indicar os dígitos que formam os números. Porém, há uma restrição: você não pode inserir nenhum caractere que não seja um dígito dentro do número. Por exemplo, se você quiser criar uma variável “x” e nela armazenar número um milhão cento e quinze mil e trezentos, **não poderá** utilizar ponto ou espaço para separar os milhares, isto é, não poderá fazer: x = 1.115.300 ou



Escola Nacional de Ciências Estatísticas

`x = 1 115 300`. Isto é proibido! A única coisa que o Python permite é utilizar o underscore no literal. Sendo assim, você pode fazer a declaração usando `x = 1115300` (mais comum) ou `x = 11_115_300` (menos usado).

E como podemos declarar **valores negativos**? É simples, basta usar o sinal de menos, por exemplo, `z = -99`.

Também é possível especificar números no formato binário, octal e hexadecimal. Para um número binário, devemos prefixar o literal com “0b”. Por exemplo `n1 = 0b10` declara a variável “n1” com o valor binário 10, que é equivalente ao valor decimal 2. Para especificar um número octal o prefixo é “0o”. Desta forma, `n2 = 0o123` declara a variável “n2” com o valor octal 123, que corresponde ao valor decimal 83. Por fim, “0x” permite a especificação de números no formato hexadecimal. Assim, `n3 = 0xA3F` ou `n3 = 0xa3f` são comandos de atribuição para declarar a variável “n3” com o valor hexadecimal A3F, que corresponde ao valor decimal 2623. É importante deixar claro que mesmo que os valores das variáveis sejam atribuídos com esses prefixos, estes valores serão sempre exibidos no formato decimal quando você usar a função `print()`.

(c) tipo **float**:

Pode armazenar um número real (fracionário), seja ele negativo, nulo ou positivo. Exemplos: 0, 1, -1, 3.14, -99.99, 1.50, 1000, 100000. Veja que o ponto “.” é utilizado para definir a parte decimal, e não a vírgula “,” – as linguagens de programação seguem o padrão do idioma inglês.

Também é possível utilizar a notação científica para declarar variáveis do tipo float, bastando para isso separar a base e o expoente com o uso da letra “E” (ou “e”). Por exemplo, `y = 3E8` declara a variável float “y” com o valor $3.0 \times 10^8 = 300000000.0$. Outro exemplo interessante é a declaração `h = 6.62607015E-34`, que declara a variável float “h” com o valor da Constante de Planck (https://pt.wikipedia.org/wiki/Constante_de_Planck), que é igual a $6.62607015 \times 10^{-34}$.

(d) tipo **bool**

Em Português é chamado de **tipo lógico**. Armazena apenas um destes dois valores: True (verdadeiro) ou False (falso). Este tipo de variável deve ser preferencialmente utilizado quando desejamos armazenar uma informação cujo valor será ou verdadeiro ou falso. Por exemplo: Possui Carro? Possui Casa Própria? Tem cartão de crédito? Foi aprovado na disciplina?.

Embora simples, trata-se de um tipo de dado muito importante na programação. Você irá lidar com ele inúmeras vezes ao longo deste curso.

IV.3.3 Operações Aritméticas

Os programas de computador são comumente empregados para a realização de cálculos. A tabela abaixo apresenta os principais operadores aritméticos do Python. Estes operadores permitem a realização de cálculos sobre números inteiros (int) ou reais (float).



Escola Nacional de Ciências Estatísticas

Operadores Aritméticos

Operador	Operando	Resultado
+ (adição)	Inteiro ou real	Inteiro ou real
- (subtração)	Inteiro ou real	Inteiro ou real
* (multiplicação)	Inteiro ou real	Inteiro ou real
/ (divisão real)	Inteiro ou real	Real
// (quociente da divisão)	Inteiro ou real	Inteiro ou real
% (resto da divisão)	Inteiro ou real	Inteiro ou real
** (exponenciação)	Inteiro ou real	Inteiro ou real

O programa a seguir apresenta exemplos de utilização destes operadores, tornando mais claros os conceitos apresentados.

```
#P002: operações aritméticas básicas
i1 = 6 + 1 * 10
i2 = (6 + 1) * 10
i3 = i1 + i2
i4 = 15 // 2
i5 = 15 % 2
r1 = 10 - 4 / 2
r2 = -2.5 ** 3

print(i1)
print(i2)
print(i3)
print(i4)
print(i5)
print(r1)
print(r2)
```

```
>>>
16
70
86
7
1
8.0
-15.625
```

Agora uma rápida explicação sobre o programa. Na linha 2, a variável “i1” recebe o valor 16 como conteúdo. As operações de multiplicação e divisão possuem prioridade sobre a soma e a subtração, mas esta prioridade pode ser modificada com o uso de parênteses, como ocorre na linha 3 do programa (o valor atribuído a “i2” é 70).

A instrução da linha 4 faz com que “i3” receba a soma do conteúdo de “i1” (16) e de “i2” (70). Ou seja, atribui-se 86 para a variável “i3”.

As linhas 5 e 6 demonstram os operadores // (divisão inteira) e % (resto da divisão). Estes operadores são normalmente aplicados a números inteiros, embora também seja possível fazê-los operar sobre números reais no Python. 15 // 2, significa “obtenha o quociente da divisão de 15 por 2”. Com isso, a variável “i4” recebe o valor 7. Já 15 % 2



Escola Nacional de Ciências Estatísticas

significa “obtenha o resto da divisão de 15 por 2”, o que resulta em 1. Desta forma, 1 é atribuído como valor para a variável “i5”.

Na linha 7, apresenta-se uma conta envolvendo o operador “/” (divisão real). No Python, o resultado de qualquer operação envolvendo “/” resulta em um número real e por isso será armazenado numa variável real. A conta $10 - 4 / 2$ resulta em 8.0 e este valor é armazenado em “r1”. O resultado é inteiro, mas por padrão qualquer operação que envolva a divisão de dois números inteiros no Python sempre resultará em um valor do tipo real (float). Na oitava linha do programa apresenta-se um exemplo de operação de exponenciação, onde o valor negativo -2.5 é elevado ao cubo, resultando em -15.625. O programa finaliza com uma série de chamadas à função `print()` para que o conteúdo de cada uma das variáveis seja exibido na saída para o usuário.

A tabela a seguir apresenta a ordem de prioridade dos operadores durante a resolução de operações aritméticas. Nenhuma surpresa: exponenciação tem a maior prioridade. Em seguida, multiplicação, divisão real, divisão inteira e resto da divisão possuem a mesma prioridade. Por fim, adição e subtração têm a menor prioridade.

prioridade	operadores
1	**
2	*, /, //, %
3	+, -

Sendo assim $2 + 5 * 3 ** 2$ resulta em 47, já que primeiro realiza-se a exponenciação, seguida da multiplicação, seguida da adição. Você pode modificar as prioridades utilizando parênteses, como em $((2 + 5) * 3) ** 2$ cujo resultado é 441. Colchetes e chaves não são aceitos para definir prioridades de operações. Você pode utilizar apenas parênteses.

Em expressões onde todos os operadores possuem a mesma prioridade, os cálculos são conduzidos **da esquerda para direita**. Por exemplo, na expressão $5 * 4 \% 3$, primeiro calcula-se $5 * 4$, que resulta em 20. Então calcula-se $20 \% 3$, que resulta em 2, valor final da expressão.

O **único operador que trabalha de forma diferente** é o operador de exponenciação, que realiza os cálculos da direita para a esquerda. Por exemplo, considere 2^{2^3} , que pode ser expresso no Python como $2 ** 2 ** 3$. Nesse caso, o primeiro cálculo a ser realizado será $2 ** 3$, resultando em 8. Então será feito $2 ** 8$, resultando em 256. Note que, neste caso, se o Python calculasse considerando a ordem da esquerda para direita, o valor final do cálculo ficaria errado (daria 64 em vez de 256).

IV.3.5 Operações com String: Concatenação e Repetição

Os operadores + e * também podem ser utilizados sobre strings. Porém, para este tipo de dado o símbolo + é usado para concatenar (o que significa juntar duas strings) enquanto o * para repetir (repetir uma string n vezes). Veja o exemplo abaixo:

```
#P003: concatenação e repetição de strings
a = 'ENCE'
b = 'Estatística'
```




```
#concatenação
print(a + " --> " + b)

#repetição
print(a * 5)
```

```
>>>
ENCE --> Estatística
ENCEENCEENCEENCEENCE
```

Este programa declara duas variáveis “a”, “b”, ambas do tipo string, a primeira com o valor ‘ENCE’ e a segunda com o valor ‘Estatística’. Em seguida, a instrução `a + " --> " + b` é usada dentro da função `print()`. Nesta instrução, o operador de concatenação `+` é usado para concatenar o valor da variável “a” com o literal “-->” e o valor da variável “b”, resultando na string “ENCE --> Estatística”.

No mesmo programa, a instrução `a * 5` utiliza o operador de repetição `*` para repetir o conteúdo de “a” 5 vezes, resultando na string “ENCEENCEENCEENCEENCE”.

IV.4 Entrada de dados

Nas linguagens de programação, uma **instrução de entrada** serve para permitir com que o programa possa **receber dados** do ambiente externo (dados especificados pelo usuário) e armazená-los em variáveis. A linguagem Python possui uma função que permite a entrada via teclado: **`input()`**.

A instrução de entrada é aquela que permite que o usuário digite dados, possibilitando um “diálogo com o computador”. **O dado digitado é copiado imediatamente para a variável especificada.** Observe o seguinte exemplo:

```
nome = input()
```

Veja que se trata de um comando de atribuição. Este comando significa: “leia um valor do teclado e armazene na variável nome”. Quando o computador “vê” a função `input()` do lado direito da atribuição, ele fica esperando o usuário digitar uma informação. O computador não passa para a instrução seguinte do programa enquanto o usuário não digitar uma informação e teclar ENTER. Após o usuário digitar algo e teclar ENTER, a informação digitada será armazenada na variável que for especificada do lado direito. Veja o exemplo a seguir:

```
#P004: entrada de dados básica
print('Qual o seu nome?')
nome = input()
print('Hmmm... então você é o famoso', nome)
```

```
>>>
Qual o seu nome?
```



Escola Nacional de Ciências Estatísticas

Nelson Cavaquinho

Hmmm... então você é o famoso Nelson Cavaquinho

A função `input()` também pode ser utilizada com um argumento, para que seja possível fazer uma pergunta ao usuário sem utilizar a função `print()`. Veja o exemplo abaixo:

```
#P005: entrada de dados básica - usando o input com parâmetro
nome = input('Qual o seu nome?')
print('Hmmm... então você é o famoso', nome)
```

```
>>>
Qual o seu nome? Tom Jobim
Hmmm... então você é o famoso Tom Jobim
```

**** IMPORTANTE:** agora precisamos esclarecer uma coisa que **você não pode esquecer**: o resultado retornado pela função `input()` será sempre uma string. Essa string conterá os caracteres que o usuário digitar do teclado. Mesmo que o usuário digite apenas números, o valor será retornado pelo `input()` como uma string. Logo a variável que for usada em conjunto com o `input()` no lado esquerdo da atribuição será uma variável do tipo `str`. Veja o exemplo a seguir:

```
#P006: o input() sempre gera uma string !!!
n = input('Digite um número: ')
print('O triplo do número é:', n * 3)
```

```
>>>
Digite um número: 2
O triplo do número é: 222
```

Neste exemplo, o usuário digitou o valor 2. Porém, como o `input()` sempre retorna os dados digitados como uma string, o valor armazenado na variável “n” foi a string ‘2’ e não o inteiro 2. Na linha seguinte, ao fazer `n * 3`, como “n” é do tipo string, na verdade o Python não multiplica 2 por 3, mas sim, faz a repetição da string 3 vezes, o que resulta em ‘222’.

Felizmente, a linguagem Python oferece duas funções simples para resolver esse problema: `int()` and `float()`.

- A função `int()` recebe uma informação como entrada (em nosso caso, uma string) e tenta convertê-la para o tipo inteiro caso isso seja possível. Se a conversão falhar, todo o programa também falhará – mais precisamente, uma mensagem de erro será exibida e o programa abortará (será encerrado).
- A função `float()` funciona da mesma maneira, porém converte a informação de entrada para o tipo float.

No exemplo a seguir, a função `float()` é utilizada no programa que calcula o triplo de um número lido do teclado, com o objetivo de converter o valor digitado para o tipo `float()`. Para um exemplo envolvendo a função `int()`, consulte a Unidade 1, seção I.4.



Escola Nacional de Ciências Estatísticas

```
#P007: convertendo um valor lido para float()
n = float(input('Digite um número: '))
print('O triplo do número é:', n * 3)
```

```
>>>
Digite um número: 2
O triplo do número é: 6.0
```

Neste programa, a instrução `n = float(input('Digite um número: '))` significa: “aguarde o usuário digitar algo e depois que ele tiver digitado, converta a informação digitada é um número real” e depois armazene-a na variável “n”. No exemplo apresentado, veja que o usuário digitou ‘2’, que com `float()` foi convertido para 2.0. O resultado de `2.0 * 3` é 6.0 (um valor real), que é apresentado na saída para o usuário.

Um erro ocorrerá, caso o usuário digite um valor que não possa ser convertido para um número real, como “ENCE” ou “Estatística”, por exemplo. Em unidades futuras, aprenderemos a “defender” o nosso programa deste tipo de entrada. Aguarde!

IV.5 Saída de dados

Uma **instrução de saída** é responsável por **enviar dados** (um resultado ou uma mensagem) para o usuário, através de um dispositivo de saída (monitor, impressora, etc.). A linguagem Python possui uma função que realiza a saída em vídeo: **print()**.

A informação a ser exibida como saída deve estar especificada entre parênteses. Através de uma instrução de saída é possível escrever três tipos de informação na tela: uma mensagem, o valor de uma variável ou o resultado de uma operação aritmética.

IV.5.1 Escrevendo uma mensagem

Para escrever uma mensagem na tela, basta usar a função `print()` e especificar a mensagem **entre aspas simples ou duplas**.

```
#P008: escrevendo uma mensagem
print('Olá')
print("ENCE")
print("Rio de Janeiro")
```

```
>>>
Olá
ENCE
Rio de Janeiro
```

Este programa não possui entrada e não realiza nenhum processamento. Somente escreve três mensagens na tela. Veja que, por padrão, o `print()` imprime a informação desejada e depois força uma quebra de linha. Por isso, cada mensagem é impressa em uma linha diferente.

IV.5.2 Escrevendo o valor de uma variável



Escola Nacional de Ciências Estatísticas

Para escrever o valor de uma variável, é necessário especificar a variável sem usar aspas simples. Veja o exemplo abaixo: o que aparece na tela é o valor de x (10).

```
#P009: escrevendo valor de variável
x = 10
print(x)

>>>
10
```

IV.5.3 Escrevendo o valor de uma expressão

Para escrever o valor de uma expressão, basta introduzi-la sem usar aspas (aspas são usadas apenas quando desejamos imprimir mensagens). No exemplo seguinte, o Python calcula a expressão $x * 10 + 100$ (resulta em 200, pois x é igual a 10) e, em seguida, exibe o seu resultado na tela.

```
#P009: escrevendo expressão
x = 10
print(x * 10 + 100)

>>>
200
```

IV.5.4 Escrevendo mais de uma informação

Conforme visto em exemplos anteriores, é possível utilizar um mesmo comando `print()` para escrever várias coisas, bastando **separá-las por vírgula**. No exemplo abaixo, um mesmo comando `print()` é usado para escrever uma mensagem, o valor de uma variável e o resultado de um cálculo. Um espaço em branco será utilizado para separar cada informação no momento em que forem exibidas na tela.

```
#P010: escrevendo várias coisas
x = 10
print('o cubo de', x, 'é', x ** 3)

>>>
o cubo de 10 é 1000
```

IV.5.5 Parâmetros `sep` e `end`

A função `print()` possui dois parâmetros opcionais descritos a seguir:

- `sep`: quando usamos o `print()` para escrever várias informações, por padrão elas serão apresentadas separadas por espaço. Com o uso do parâmetro `sep`, você poderá especificar outro caractere qualquer.
- `end`: por padrão, a função `print()` imprime uma mensagem e em seguida gera uma quebra linha. Com o uso do parâmetro `end`, você poderá fazer com que todo comando `print()` não encerre com uma quebra de linha, mas com o uso de algum caractere, como espaço em branco por exemplo.



Escola Nacional de Ciências Estatísticas

Para deixar tudo mais claro, veja a forma de utilizar esses parâmetros no exemplo a seguir.

```
#P011: print() - parâmetros sep e end
a=1; b=2; c=3

#imprime os valores de "a", "b" e "c" separados por espaço
print(a,b,c) #1 2 3

#parâmetro "sep": troca espaço pelo separador especificado, neste caso
#                ponto-e-vírgula
print(a,b,c,sep=';') #1;2;3

#parâmetro "end": por padrão, a função termina uma linha com
#quebra de linha. Mas você pode terminar com qualquer caractere usando
#o parâmetro "end". No exemplo abaixo, usamos espaço
print("Ciência",end=' ')
print("de",end=' ')
print("Dados")
```

```
>>>
1 2 3
1;2;3
Ciência de Dados
```

Obs1.: veja que a primeira linha contém três instruções de atribuição separadas por ponto-e-vírgula. Isso é permitido no Python, você pode definir mais de um comando em uma única linha, contanto que os separe por ponto-e-vírgula.

Obs2.: Em uma unidade posterior, revisitaremos a função print() para apresentar suas opções de formatação de valores.

IV.6 Problemas resolvidos

Esta seção apresenta uma série de exercícios resolvidos que exemplificam a utilização de variáveis e os comandos de entrada e saída (estes são utilizados de diferentes maneiras). Para observar o funcionamento dos programas, digite e execute os exemplos no Thonny ou outro ambiente Python.

EXEMPLO IV.6.1 Escreva um programa que solicite ao usuário um valor de distância em MILHAS. Em seguida calcule e imprima o valor equivalente em QUILOMETROS.

RESOLUÇÃO:

```
print('* * Programa Conversor: Milhas -> Km * *')
print()
milhas = float(input('Digite o valor da distancia em Milhas: '))
```



Escola Nacional de Ciências Estatísticas

```
km = milhas * 1.61
print()
print('O valor equivalente em km é:', km)
```

EXPLICAÇÃO:

- A primeira linha exibe uma mensagem que informa ao usuário sobre o propósito do programa.
- A instrução da linha 2 (**print** sem nenhum parâmetro) é usada apenas por motivos “estéticos”. Ele faz com que uma linha em branco seja impressa na tela.
- O comando da linha 3 é um comando de atribuição que contém a função **input()**, que realiza a entrada no Python. Quando o computador “vê” essa instrução ele “sabe” que deve esperar o usuário digitar um valor no teclado. A informação digitada será convertida para um valor do tipo float (com o uso da função **float()**) e depois armazenada na variável “milhas”.
- A linha 4 um comando que converte o valor da distância em milhas para quilômetros, através de uma operação aritmética. O resultado é armazenado na variável “km”, com o uso do operador de atribuição “=”.
- A instrução da linha 5 imprime uma linha em branco na tela.
- Por fim, a linha 6 imprime uma mensagem e o valor armazenado na variável “km”. Note que a mensagem é especificada entre aspas, depois usa-se o caractere vírgula e o identificador “km” (a vírgula é utilizada para separar as coisas que desejamos imprimir no **print()**).

Uma maneira mais “bonita” de resolver este programa é alterar a linha 11 da seguinte forma:

```
print('O valor equivalente em km eh: ', round(km, 2))
```

Esta alteração utiliza a função **round()** para possibilitar com que o valor da variável “km” seja arredondado e consequentemente impresso com 2 casas decimais.

RESOLUÇÃO ALTERNATIVA:

```
print('* * Programa Conversor: Milhas -> Km * *')
print()
milhas = float(input('Digite o valor da distancia em Milhas: '))
print()
print('O valor equivalente em km é:', milhas * 1.61)
```

EXPLICAÇÃO:



Escola Nacional de Ciências Estatísticas

Neste exemplo, optamos por não declarar a variável “km”. O valor da conversão de milhas para quilômetros é calculado e impresso dentro do próprio comando print().

EXEMPLO IV.6.2 A conversão de graus Celsius para graus Fahrenheit é obtida pela seguinte fórmula:

$$F = C \times 1,8 + 32$$

Elabore um programa que receba como entrada o valor da temperatura em graus Celsius e imprima o valor da temperatura em graus Fahrenheit.

RESOLUÇÃO:

```
print('--> Programa Conversor: Celsius -> Fahrenheit')
print()
c = float(input('Digite o valor da temperatura em graus Celsius: '))
f = c * 1.8 + 32
print()
print('Valor equivalente em Fahrenheit = ', round(f, 2))
```

Obs.: round(f, 2) faz com que o valor de “f” seja arredondado para 2 casas decimais.

EXEMPLO IV.6.3 A Escreva um programa que solicite ao usuário as seguintes informações: distância a ser percorrida (em Km) e consumo de combustível do carro (em Km / litros).

Em seguida o programa deve calcular e imprimir quantos litros de combustível são necessários para percorrer a distância.

RESOLUÇÃO:

```
print('* * Programa LITROS DE GASOLINA * *'); print()
print('Digite o valor da distancia que voce vai percorrer (em Km): ')
dist = float(input())
print('Especifique o consumo do seu carro (em Km/l): ')
consumo = float(input())
litros = dist / consumo
print('Voce deve abastecer ', litros, ' litros de combustivel')
```

EXEMPLO IV.6.4 Escreva o seguinte programa “interativo” em Python: o computador deve fazer a seguinte pergunta ao usuário “Qual o seu nome?”.

O usuário então, digita seu nome e, em seguida, o computador deve imprimir a seguinte mensagem na tela:

Bom dia, fulano!

Você já está ficando “fera” em Python...



RESOLUÇÃO:

```
print('Qual o seu nome?')
nome_usuario = input()
print()
print('Bom dia', nome_usuario, '!')
print('Voce ja está ficando "fera" em Python... ')
```

EXEMPLO IV.6.5 Exemplo de programa que demonstra a alteração em conteúdo de variável ao longo do bloco de comandos.

RESOLUÇÃO:

```
nome = 'ENCE'
print(nome)
nome = 'IBGE'
print(nome)
```

```
>>>
ENCE
IBGE
```

EXEMPLO IV.6.6 Escrever um programa que funcione da seguinte maneira:

Inicialmente, o programa deve ler um número **inteiro** digitado pelo usuário e armazená-lo numa variável chamada x.

Em seguida, o sistema deve calcular e imprimir o valor da função $y = f(x) + g(x)$, onde:

$$h(x) = x^2 - 16$$

$$f(x) = h(x) \times 2;$$

$$g(x) = h(x) + 4;$$

RESOLUÇÃO:

```
print('programa y = f(x) + g(x)')
print()
x = int(input('Digite o valor de x: '))
hx = (x * x) - 16
fx = hx * 2
gx = hx + 4
y = fx + gx
print()
print('y = ', y)
```




Escola Nacional de Ciências Estatísticas

EXEMPLO IV.6.7 Escreva um programa que receba como entrada o valor do raio de uma esfera e calcule e imprima o valor da área e do volume.

$$A = 4\pi r^2$$

$$V = 4\pi r^3 \div 3$$

RESOLUÇÃO:

```
print('programa que calcula a area e volume da esfera')
print()
PI = 3.14
raio = float(input('Digite o valor do raio: '))
area = 4 * PI * (raio ** 2)
volume = 4/3 * PI * (raio ** 3)
print('Area =', area)
print('Volume =', volume)
```

IV.7 Funções matemáticas – módulo ‘math’

O Python possui uma biblioteca de módulos que podem ser utilizados em programas para que novas funcionalidades fiquem disponibilizadas. Um deles é o módulo ‘math’ que fornece uma série de funções e constantes matemáticas úteis, tais como funções de arredondamento, trigonométricas, logarítmicas, etc. Antes de utilizá-lo, você precisa realizar a sua **importação** da seguinte maneira:

```
import math
```

Na unidade sobre funções, forneceremos mais detalhes sobre módulos e o comando import. Por enquanto, apenas entenda que sempre que você precisar utilizar algum recurso do módulo math, é preciso usar o import acima. A seguir, são relacionadas as principais constantes e funções deste módulo. Você não precisa decorar essas constantes e funções, mas deve saber como utilizá-las em seus programas.

Constantes do módulo 'math'

- `math.pi`: constante matemática 3.14159...;
- `math.e`: constante matemática 2.718281... (número de Euler);
- `math.tau`: constante matemática 6.283185... (equivalente à 2π);
- `math.inf`: valor float que representa $+\infty$. Para $-\infty$ basta usar `-math.inf`;
- `math.nan`: trata-se do famoso NaN, valor float que representa “not a number”. Este valor é gerado pelo Python sempre que o resultado de um cálculo não puder ser expresso por um número. Qualquer cálculo envolvendo NaN sempre resultará em NaN (ex.: $1 + \text{NaN} = \text{NaN}$).



Funções do módulo 'math'

Funções de Arredondamento

- `math.ceil(x)`: arredondamento “pra cima”, ou seja, retorna o menor inteiro com valor igual ou superior a x ;
- `math.floor(x)`: arredondamento “pra baixo”, ou seja, retorna o maior inteiro com valor igual ou inferior a x .
- `math.trunc(x)`: truncamento, o que significa limitar o número de dígitos de x .

Funções Logarítmicas / Exponenciais

- `math.exp(x)`: retorna e elevado a x ;
- `math.log2(x)`: retorna o logaritmo de x na base 2;
- `math.log10(x)`: retorna o logaritmo de x na base 10;
- `math.log(x, b)`: retorna o logaritmo de x na base b (de acordo com a documentação do Python, deve ser utilizada apenas quando b é diferente de 2 e 10).
- `math.pow(x, y)`: retorna x elevado a y ;
- `math.sqrt(x)`: retorna a raiz quadrada de x ;

Funções Trigonômicas (em todas elas, x é um ângulo que deve ser fornecido em radianos)

- `math.acos(x)`: retorna o arco cosseno de x ;
- `math.asin(x)`: retorna o arco seno de x ;
- `math.atan(x)`: retorna o arco tangente de x ;
- `math.cos(x)`: retorna o cosseno de x ;
- `math.sin(x)`: retorna o seno de x ;
- `math.tan(x)`: retorna a tangente de x ;
- `math.degrees(x)`: converte o ângulo x de radianos para graus;
- `math.radians(g)`: converte o ângulo g de graus para radianos.

Funções Hiperbólicas

- Também existem as funções hiperbólicas análogas, cujos os nomes sempre terminam com letra “h”. Ex.: `math.tanh(x)` retorna a tangente hiperbólica de x .

Funções para Teste de Valores

- `math.isnan(x)`: retorna `True` caso x seja `NaN`; caso contrário `False` é retornado;
- `math.isfinite(x)`: retorna `True` caso x não seja infinito e nem `NaN`; caso contrário, `False` é retornado;
- `math.isinf(x)`: retorna `True` caso x seja infinito (positivo ou negativo); caso contrário, `False` é retornado,



Escola Nacional de Ciências Estatísticas

A seguir, um programa que utiliza recursos do módulo `math`. Observe que para utilizar uma função ou constante deste módulo, é preciso prefixar seu nome com “`math.`” (ou seja, o nome do módulo e um ponto antes do nome da função ou constante).

```
#P012: módulo 'math'
import math

#constante PI
print('PI=',math.pi) #3.141592653589793

#funções de arredondamento
x1 = 5.9
print()
print(x1)
print('ceil',math.ceil(x1))
print('floor',math.floor(x1))
print('trunc',math.trunc(x1))

#logaritmo
print()
x2 = 1024
print('log de',x2,'na base 2:', math.log2(x2))

#raiz quadrada
x3 = 81
print('raiz quadrada de',x3,':', math.sqrt(x3))

#imprime tabela com seno, cosseno e tangente de 30, 45 e 60
#note que é preciso converter os ângulos para radianos
print()
angulo_graus = 30
angulo_radianos = math.radians(angulo_graus)
print('\n* * * Angulo=',angulo_graus, ' graus')
print('SENO =',round(math.sin(angulo_radianos),2))
print('COSSENO =',round(math.cos(angulo_radianos),2))
print('TANGENTE =',round(math.tan(angulo_radianos),2))
```

>>>

PI= 3.141592653589793

5.9

ceil 6

floor 5

trunc 5

log de 1024 na base 2: 10.0

raiz quadrada de 81 : 9.0

* * * Angulo= 30 graus

SENO = 0.5



Escola Nacional de Ciências Estatísticas

COSSENO = 0.87
TANGENTE = 0.58

Exercícios propostos

- (1) Faça um programa que solicite a idade de uma pessoa e que, depois, imprima quantos anos ela terá em 2050.
- (2) Faça um programa que receba a base e a altura de um triângulo e que calcule e imprima a sua área.
- (3) Escreva um programa capaz de gerar a tela apresentada a seguir:

```
>>> %Run e3.py

*****  *   *  *****  *****
*       **  *   *       *
***     * * *   *       ***
*       *   **  *       *
*****  *   *  *****  *****
```

- (4) Quais serão os valores das variáveis x1, x2, y1, y2 ao final da execução do programa abaixo?

```
import math
x1=0; x2=2; y1= 9 ** 2
y2 = (y1 % 5) + (y1 // 9)
y1 = math.trunc(8.9) + round(9.1)
x1 = 5 * 10 + 12 / 3;
x1 = y1 * 10
```

- (5) Escreva um programa que receba como entrada o valor de um número inteiro N e que calcule o valor de S da seguinte forma:

$$S = 1 * (N - 1) + 2 * (N - 2) + 3 * (N - 3) + 4 * (N - 4) + 5 * (N - 5)$$

- (6) Escreva um programa para o cálculo do valor de uma prestação em atraso, utilizando a fórmula: prestação = valor_original + (valor_original * (taxa / 100) * tempo). Nesta fórmula, o “tempo” é dado em meses; “valor_original” representa o valor original da prestação, dado em atraso; “taxa” é um valor entre 0 e 100. Apresente o resultado calculado para o usuário com o valor arredondado para 2 casas decimais.

Importante: observe a fórmula acima e decida que dados você precisará receber como entrada.



Escola Nacional de Ciências Estatísticas

(7) Faça um programa que receba como entrada o nome, a idade e o valor do salário mensal de uma pessoa. Em seguida calcule o salário desta pessoa em quantidade de salários mínimos e imprima a seguinte mensagem na tela:

Seu nome é *fulano*, você tem x anos e você recebe z salários mínimos por mês.

(8) Faça um programa que receba dois números inteiros e imprima:

- o quadrado da diferença do primeiro valor pelo segundo;
- a diferença dos quadrados dos dois números

(9) Faça um programa que receba o valor de x como entrada e calcule y utilizando a seguinte função:

$$y = -\log_2 x$$

(10) Faça um programa que o valor de um ângulo em graus e calcule e exiba o seno, cosseno e a tangente desse ângulo.

(11) Faça um programa que receba 10 números reais como entrada e calcule e imprima a média destes 10 números.

(12) Considere uma empresa de engenharia que possui um programa para a contratação de recém-formados. Neste programa, 10 engenheiros “*trainees*” são contratados anualmente. Estes novos funcionários começam a trabalhar no mês de janeiro, com salário inicial de R\$ 2.000,00. No mês de junho os *trainees* recebem um aumento salarial de 2%. No mês de Dezembro o salário é aumentado em mais 5%. Elabore um programa que calcule e imprima o valor gasto pela empresa **ao final do ano** com os salários dos 10 engenheiros *trainees* (para simplificar, desconsidere o 13º salário).

(13) Escreva um programa que leia o valor de duas variáveis “a” e “b” do tipo **string**, efetue a troca dos valores, de forma que a variável “a” passe a ter o valor da variável “b” e que a variável “b” passe a ter o conteúdo da variável “a”. Apresente na tela os valores trocados.

(14) Faça um programa que leia um número inteiro de 4 dígitos e que imprima, em duas linhas separadas, os dois primeiros e os dois últimos algarismos deste número:

Ex: se usuário digitar o número 3025, o programa deve imprimir:

- Dois primeiros algarismos = 30
- Dois últimos algarismos = 25

Dica: utilize os operadores // e %