

Python Básico

Marcos Roberto Ribeiro

**Formação Inicial e
Continuada**

+ IFMG

Campus Bambuí



Objetivos

- Conhecer os tipos básicos de dados e funcionamento de variáveis;
- Desenvolver códigos com entrada e saída de dados;
- Entender o funcionamento de expressões aritméticas, relacionais, lógicas e suas combinações



Mídia digital: Antes de iniciar os estudos, vá até a sala virtual e assista ao vídeo “Apresentação do curso”.

1.1 Introdução

As linguagens de programação são ferramentas essenciais para o desenvolvimento de softwares. Uma das linguagens de programação mais usadas no mundo é a linguagem Python¹ (TIOBE, 2021; PYPL, 2021). Devido a grande popularidade do Python, seu aprendizado por parte dos desenvolvedores pode ser um grande diferencial no mercado de trabalho. Além disso, desde sua criação no final da década de 1980, a linguagem passou por uma constante evolução até chegar à versão 3 em 2008 (WIKIPÉDIA, 2021b). A linguagem Python é uma linguagem aberta, multiplataforma e utilizada uma grande variedade de aplicações (MENEZES, 2019; RAMALHO, 2015).

1.1.1 Lógica de programação

Ao mesmo tempo que desenvolvemos códigos utilizando uma linguagem Python, precisamos entender a lógica de programação por trás desses códigos. No estudo da lógica de programação, normalmente, utilizamos algum tipo de pseudocódigo para escrever algoritmos. Os algoritmos são sequências finitas de instruções para resolver determinados problemas. Apesar de não parecer, podemos descrever na forma de algoritmos várias tarefas do nosso, como, por exemplo, fazer uma receita culinária ou analisar se um aluno foi aprovado. Basicamente, um algoritmo recebe uma entrada, efetua um processamento e retorna um resultado. No caso de uma receita culinária, os ingredientes são a entrada, a execução da receita é o processamento, e o prato pronto é o resultado.

Os algoritmos podem ser representados de diversas maneiras, como descrição narrativa, fluxogramas ou códigos Python. Assim, para escrever um algoritmo corretamente é preciso entender a sintaxe e a semântica da linguagem. A sintaxe define as regras de

¹ <https://www.python.org/>

escrita, ou seja, se determinada instrução está escrita corretamente. Por outro lado, a semântica é o significado da instrução.

Na descrição narrativa os algoritmos são escritos com o mínimo de formalidade usando a linguagem natural (GOMES; BARROS, 2015). Apesar de não haver grande formalidade, a descrição narrativa deve ser uma sequência de frases curtas, claras e objetivas. É recomendável usar apenas um verbo por frase. A Figura 1 apresenta um exemplo de algoritmo na forma de descrição narrativa para verificar a aprovação de um aluno considerando a médias de duas notas. Contudo, a descrição narrativa não é muito usada na prática. Isso acontece porque a linguagem natural pode gerar ambiguidade e sua execução por computador é muito mais complexa do que a execução de instruções em linguagens formais.

- 1) Somar as duas notas do aluno
- 2) Calcular a nota final dividindo a soma por dois
- 3) Se a nota final for maior que 60, o aluno foi aprovado
- 4) Senão, o aluno foi reprovado

Figura 1 – Algoritmo para verificar aprovação de aluno em descrição narrativa
Fonte: Elaborado pelo Autor.

Os fluxogramas usam símbolos específicos para as ações dos algoritmos e setas para conectar esses símbolos. A Figura 2 mostra os símbolos usados em fluxogramas bem como seus respectivos significados.

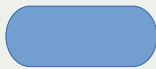





Símbolo	Significado
	Terminador (início e fim do algoritmo)
	Leitura (entrada de dados)
	Processo (processamento de dados)
	Escrita (saída de dados)
	Decisão (desvio do fluxo de execução)
	Conector (junção de fluxos de execução)

Figura 2 – Símbolos usados em fluxogramas
Fonte: Elaborado pelo Autor.

A Figura 3 apresenta novamente o algoritmo para verificar a aprovação de um aluno através de um fluxograma. Podemos observar que no fluxograma o algoritmo requer mais rigor formal do que na descrição narrativa. Após o *início*, nos paralelogramos, ocorre a entrada de dados das notas do aluno ($N1$ e $N2$). Em seguida, no retângulo, a média (M) é calculada com a expressão $M \leftarrow (N1 + N2)/2$. O próximo passo é o losango que verifica se a média está acima de 60. Em caso afirmativo, a mensagem *Aprovado* é apresentado. Caso contrário, a mensagem *Reprovado* é mostrada. Os dois fluxos de execução se encontram no conector e o algoritmo é finalizado.

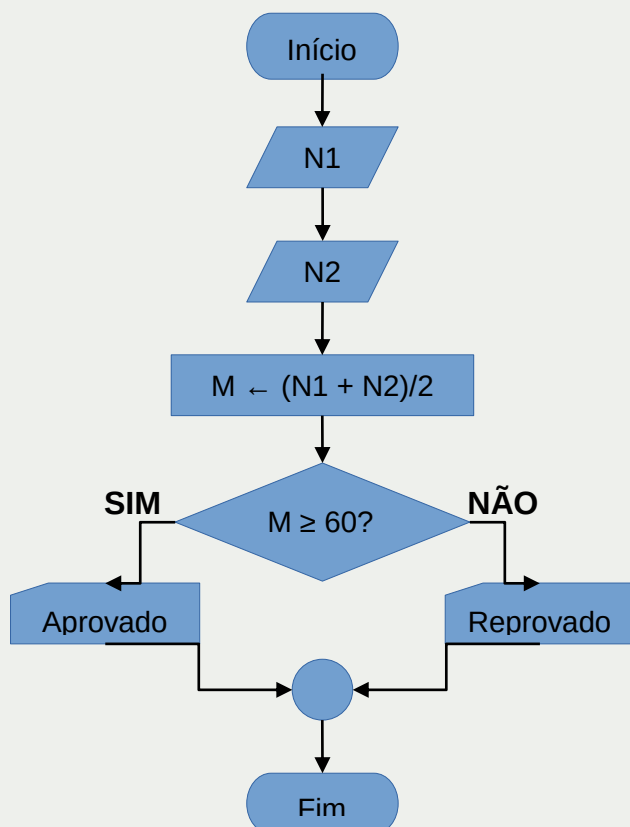


Figura 3 – Algoritmo para verificar aprovação de aluno em fluxograma

Fonte: Elaborado pelo Autor.

A Figura 4 apresenta o código para verificar a aprovação de aluno na linguagem Python. Observe que as linhas de código são numeradas para facilitar a identificação de instruções ou possíveis erros. Por exemplo, na linha 5 temos a instrução **print('Aprovado')** que escreve a mensagem de aprovado na tela. Ao longo do curso, aprenderemos outras instruções da linguagem Python.

```

1 n1 = float(input('Informe nota 1: '))
2 n2 = float(input('Informe nota 2: '))
3 m = (n1 + n2) / 2
4 if m >= 60:
5     print('Aprovado')
6 else:
7     print('Reprovado')
  
```

Figura 4 – Código Python para verificar aprovação de aluno

Fonte: Elaborado pelo Autor.

1.1.2 A ferramenta Spyder

Na maioria das distribuições do sistema operacional Linux, o ambiente Python já vem instalado por padrão. Para outros sistemas operacionais, como Windows e MacOS, é preciso baixar e instalar esse ambiente. Contudo, no presente curso, utilizaremos a ferramenta

Spyder² que já conta com o ambiente Python. No Linux, a instalação da ferramenta está disponível nos sistemas de pacotes (TAGLIAFERRI, 2018). No caso do Windows e MacOS, é recomendável utilizar a instalação *standalone* disponível no site <https://docs.spyder-ide.org/current/installation.html>. A versão padrão do ambiente Python usada pelo Spyder é a versão 3.

O Spyder é uma ferramenta livre e multiplataforma para desenvolvimento Python. A vantagem de utilizar essa ferramenta são os diversos recursos disponíveis na mesma como editor de código, console, explorador de variáveis, depurador e ajuda. O editor de código possui um explorador de código e autocomplemento. O console permite a execução de comandos Python. Com o explorador de variáveis é possível visualizar e modificar o conteúdo das variáveis. O depurador auxilia na execução passo a passo do código e o sistema de ajuda fornece documentação de comandos da linguagem.

1.2 Tipos de dados

Os códigos em Python são usados para resolver problemas do mundo real e, ao serem executados, precisam trabalhar com diversos tipos de dados. A Figura 5 apresenta os tipos de dados básicos da linguagem Python (PSF, 2021). Os tipos de dados são usados principalmente para representar variáveis e literais. Os Literais são os dados informados literalmente no código. As variáveis são explicadas na próxima seção.

Tipo	Descrição
int	Números inteiros
float	Números fracionários
bool	Valores lógicos como True (verdadeiro) ou False (falso)
str	Valores textuais

Figura 5 – Tipos básicos de dados em Python

Fonte: Elaborado pelo Autor.

1.3 Variáveis

Uma variável representa uma posição de memória no computador cujo conteúdo pode ser lido e alterado durante a execução do código (BORGES, 2010). Em Python, deferente de outras linguagens de programação, não precisamos declarar as variáveis. As variáveis são criadas automaticamente quando ocorre uma atribuição de valor com o operador de igualdade (=). A atribuição de valores a variáveis pode ser realizada com literais, outras variáveis, resultados de expressões e valores retornados por funções.

A Figura 6 mostra um exemplo de código com os quatro tipos básicos de dados. No caso dos tipos textuais, é importante colocar o valor a ser atribuído entre aspas. Aqui cabe uma observação sobre a função **print()**. Podemos colocar quantos literais ou variáveis

² <http://www.spyder-ide.org>

desejarmos separados por vírgula. A função irá escrever todos na tela. No caso das variáveis, a função escreve seu valor na tela (PILGRIM, 2009).

```
1 n1 = 10
2 n2 = 2.5
3 nome = 'José'
4 teste = True
5 print(n1, n2, nome, teste)
```

Figura 6 – Atribuição de valores a variáveis

Fonte: Elaborado pelo Autor.

Para que não ocorram erros no código, o nome de uma variável deve obedecer às seguintes regras:

- Deve obrigatoriamente começar com uma letra;
- Não deve possuir espaços em branco;
- Não pode conter símbolos especiais, exceto o sublinhado (_);
- Não deve ser uma palavra reservada (uma palavra da já existente na linguagem, como **print**, por exemplo).



Dica do Professor: A linguagem Python, assim como diversas outras linguagens, é *case sensitive*. Nessas linguagens, a mudança de maiúscula para minúscula, ou vice-versa, modifica o nome da variável (ou outro elemento). Assim, os nomes **teste** e **Teste** são considerados distintos.

1.4 Depuração de código

Alguns recursos interessantes da ferramenta Spyder são os pontos de parada, a execução passo a passo e o explorador de variáveis. Tais recursos são muito úteis para fazer a depuração do código e encontrar possíveis erros de funcionamento.

Os pontos de paradas podem ser ativados com um clique duplo sobre o número de cada linha. Aparece um pequeno círculo vermelho antes do número da linha, se o ponto de parada foi ativado. Assim, quando o código é executado no modo depurar, o Spyder interrompe a execução no ponto de parada ativado.

A execução em modo depurar é feita pelo menu *Depurar / Depurar* ou CTRL-F5 (pelo teclado). O explorador de variáveis é o painel no lado esquerdo. Quando o código for executado, você consegue ver o valor de cada variável. A execução passo a passo, no menu *Depurar / Executar linha* ou CTRL+F10, permite executar uma linha do código de cada vez. A utilização dos recursos explicados de forma conjunta pode ser muito útil para encontrar possíveis erros no funcionamento dos códigos.

A Figura 7 mostra um exemplo de depuração de código feito no Spyder. Podemos ver que foi marcado um ponto de parada na linha 5. Os valores e tipos das variáveis podem ser vistos no explorador.

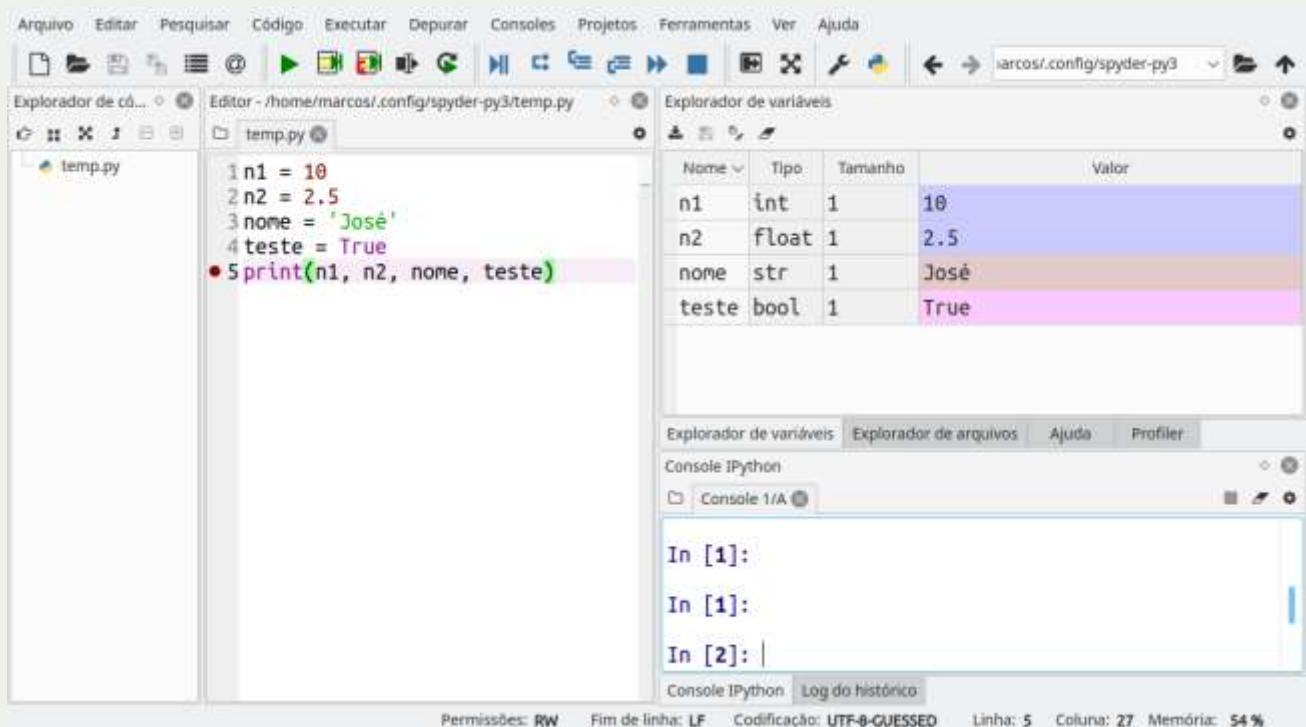


Figura 7 – Depuração de código no Spyder

Fonte: Elaborado pelo Autor.

1.5 Legibilidade

Um dos principais cuidados na escrita de códigos é a ter uma boa legibilidade. Um código escrito por você deve ser facilmente entendido por outra pessoa que vá estudá-lo ou usá-lo (ou por você mesmo no futuro). Alguns fatores que influenciam diretamente o entendimento de códigos são nomes sugestivos, endentação e comentários.

Os nomes sugestivos para variáveis e outras estruturas são importantes para que você identifique de forma rápida a referida variável ou estrutura. A endentação diz respeito aos espaços em branco a esquerda para alinhar e organizar os blocos de instruções. No caso do Python, a endentação é essencial porque é por meio dela que colocamos blocos de códigos dentro de outras instruções. Já os comentários textos não executáveis com a finalidade de explicar o código (SWEIGART, 2021). Os comentários começam com o caractere #.

```
1 # -*- coding: utf-8 -*-
2
3 n1 = 10    # Número inteiro
4 n2 = 2.5  # Número fracionário
5 nome = 'José' # Textual (sempre entre aspas)
6 teste = True # Variável lógica
```

Figura 8 – Código com comentários

Fonte: Elaborado pelo Autor.

A Figura 8 um exemplo de código com comentários em Python. Nas linhas 3 a 6, temos comentários comuns explicando as atribuições. Já na linha 1, temos um comentário especial usado para especificar a codificação de caracteres usada no arquivo. Nesse

exemplo e nos demais ao longo do livro usaremos sempre a codificação UTF8. A linguagem Python diversos outros comentários especiais usados para outras funções.

1.6 Entrada e saída de dados

Quando um algoritmo recebe dados do usuário, dizemos que ocorre uma entrada de dados. De forma análoga, quando o algoritmo exibe mensagens para o usuário, acontece uma saída de dados. Em Python, a entrada de dados é efetuada com a instrução **input()**. Dentro dos parênteses colocamos um texto para ser mostrado ao usuário antes da entrada dos dados. Normalmente, usamos uma variável para receber a resposta digitada. A função **input()** sempre retorna um valor textual digitado pelo usuário. Se precisarmos de outro tipo de dado, temos que realizar uma conversão³.

No caso da saída de dados, temos a instrução **print()**. Essa instrução recebe as informações a serem mostradas para o usuário separadas por vírgula. Se usarmos uma variável, será mostrado o valor dessa variável. Em diversas situações é importante compor mensagens adequadas para o usuário combinando literais e variáveis. A Figura 9 exibe um código usando as funções **input()** e **print()**. Na linha 5, usamos a instrução **int()** para converter texto retornado pelo **input()** para um número inteiro.

```
1 print('Bem vindo!')
2 # Nome do usuário
3 nome = input('Informe seu nome: ')
4 # Idade convertida para inteiro
5 idade = int(input('Informe sua idade: '))
6 print() # Escreve uma linha em branco
7 # Escreve mensagem usando as variáveis
8 print('Olá,', nome, 'sua idade é', idade)
9 print('Até mais!')
```

Figura 9 – Código usando **input()** e **print()**

Fonte: Elaborado pelo Autor.

A instrução **print()**, por padrão, separa os elementos recebidos com um espaço em branco e, no final, escreve o caractere de nova linha (**\n**). Esse comportamento pode ser modificado através dos parâmetros **sep** (texto para separar os elementos) e **end** (texto a ser colocado no final da linha). Por exemplo, se quisermos que os elementos fiquem colados e a saída não passe para a próxima linha, podemos usar uma instrução no seguinte formato **print(..., sep="", end="")**. As reticências (...) devem ser substituídas pelos elementos a serem escritos na tela.

1.7 Operadores

Assim como a maioria das linguagens de programação, a linguagem Python possui operadores aritméticos, relacionais e lógicos (PSF, 2021). Com esses operadores é possível

³ Podem ocorrer erros nessa conversão, mas trataremos desse detalhe.

criar expressões que também podem ser combinadas, principalmente, para a realização de testes.

1.7.1 Operadores aritméticos

A realização de cálculos matemáticos está entre as principais tarefas feitas por algoritmos. Para executarmos tais cálculos, devemos utilizar os chamados operadores aritméticos. A Figura 10 mostra os operadores aritméticos disponíveis na linguagem Python. A ordem precedência dos operadores aritméticos é a mesma ordem precedência da matemática. Também podem ser usados parênteses para alterar a ordem de precedência. A Figura 11 mostra um exemplo simples com expressões matemáticas usando os operadores aritméticos.

Operador	Descrição
+	Adição
-	Subtração
*	Multiplicação
/	Divisão
//	Divisão inteira
%	Resto de divisão
**	Potenciação

Figura 10 – Operadores aritméticos do Python

Fonte: Elaborado pelo Autor.

```

1 ni = 10 // 3
2 print('10 // 3 =', ni)
3 ni = 10 % 3
4 print('10 % 3 =', ni)
5 nr = 2.34 * 3.58
6 print('2.34 * 3.58 =', nr)
7 nr = 2.99 / 4.1
8 print('2.99 / 4.1 =', nr)
9 nr = (10.5 - 7.8) * (3.2 + 200.43)
10 print('(10.5 - 7.8) * (3.2 + 200.43) =', nr)

```

Figura 11 – Código usando operadores aritméticos

Fonte: Elaborado pelo Autor.

Com o conteúdo visto até o momento, podemos fazer um código para simular uma calculadora simples. A ideia é obter dois números com o usuário e mostrar os resultados das possíveis operações aritméticas entre esses números. A Figura 12 mostra uma possível solução.

```

1 print('Informe dois números')
2 n1 = float(input('Primeiro número: '))
3 n2 = float(input('Segundo número: '))
4 r = n1 + n2
5 print(n1, '+', n2, '=', r)
6 r = n1 - n2
7 print(n1, '-', n2, '=', r)
8 r = n1 * n2
9 print(n1, '*', n2, '=', r)
10 r = n1 / n2
11 print(n1, '/', n2, '=', r)

```

Figura 12 – Código para simular calculadora simples

Fonte: Elaborado pelo Autor.

O operador **+** pode ser usado também para concatenar variáveis e literais textuais. No entanto, não é possível concatenar diretamente um elemento textual e um elemento numérico. Nesse caso, é necessário converter o valor numérico para textual antes da concatenação. O código da Figura 13 demonstra como isso pode ser feito.

```

1 print('Informe os dados')
2 nome = input('Nome: ')
3 sobrenome = input('Sobrenome: ')
4 idade = int(input('Idade: '))
5 mensagem = nome + ' ' + sobrenome + ', ' + str(idade)
6 print(mensagem)

```

Figura 13 – Código para simular calculadora simples

Fonte: Elaborado pelo Autor.

Em Python, é comum utilizar os operadores aritméticos compostos. Eles são equivalentes à aplicação do operador seguido de uma atribuição. A Figura 14 apresenta os operadores compostos, suas formas de utilização e equivalência. Por exemplo, se temos uma variável **x** valendo **10** e usamos a expressão **x += 2**, estamos somando **2** à variável **x**, ou seja, é o mesmo que usar a expressão **x = x + 2**.

Operador	Forma de utilização	Equivalência
+=	x += y	x = x + y
-=	x -= y	x = x - y
*=	x *= y	x = x * y
/=	x /= y	x = x / y
//=	x //= y	x = x // y
%=	x %= y	x = x % y
**=	x **= y	x = x ** y

Figura 14 – Operadores compostos

Fonte: Elaborado pelo Autor.

1.7.2 Operadores relacionais

Os operadores relacionais são usados para comparar dois valores. Os valores a serem comparados podem ser literais, variáveis ou expressões matemáticas. O resultado

dessa comparação é um valor lógico **True** (verdadeiro) ou **False** (falso). A Figura 15 apresenta os operadores relacionais disponíveis na linguagem Python.

Operador	Descrição
==	Igual
!=	Diferente
<	Menor
<=	Menor ou igual
>	Maior
>=	Maior ou igual

Figura 15 – Operadores relacionais

Fonte: Elaborado pelo Autor.

O código da Figura 16 implementa um comparador de valores que possibilita testar todos os operadores relacionais para dois números informados. Execute o referido código no Spyder e observe o resultado das comparações para diferentes valores.

```

1 n1 = int(input('Informe um número: '))
2 n2 = int(input('Informe outro número: '))
3 print(n1, '==', n2, '->', n1 == n2)
4 print(n1, '!=', n2, '->', n1 != n2)
5 print(n1, '<', n2, '->', n1 < n2)
6 print(n1, '<=', n2, '->', n1 <= n2)
7 print(n1, '>', n2, '->', n1 > n2)
8 print(n1, '>=', n2, '->', n1 >= n2)

```

Figura 16 – Código comparador de valores

Fonte: Elaborado pelo Autor.

1.7.3 Operadores lógicos

Os operadores lógicos são equivalentes aos operadores da Lógica Proposicional. Em Python, temos o operador unário **not** e os operadores binários **and** e **or**. A Figura 17 mostra o funcionamento do operador **not**. Esse operador é usado para obter o oposto de um valor lógico.

Expressão	Resultado
not True	False
not False	True

Figura 17 – Operador lógico **not**

Fonte: Elaborado pelo Autor.

A Figura 18 descreve o funcionamento do operador lógico **and**. Podemos observar que esse operador retorna **True** apenas se os dois operandos forem **True**. Ou seja, se um dos operandos for **False**, o resultado do operador também será **False**.

Expressão	Resultado
True and True	True
True and False	False
False and True	False
False and False	False

Figura 18 – Operador lógico **and**

Fonte: Elaborado pelo Autor.

A Figura 19 exibe o funcionamento do operador lógico **or**. Esse operador retorna **False** somente se os dois operandos forem **False**. Como o resultado de um operador relacionado é um valor lógico, em muitas situações, os operadores lógicos são usados para combinar as comparações relacionais.

Expressão	Resultado
True or True	True
True or False	True
False or True	True
False or False	False

Figura 19 – Operador lógico **or**

Fonte: Elaborado pelo Autor.

O código da Figura 20 é um exemplo de utilização dos operadores lógicos. Utilize o Spyder para executar esse código algumas vezes informando valores diferentes para as variáveis **n1** e **n2**. Observe funcionamento do código e tente calcular sozinho os valores das variáveis **p**, **q** e **r**.

```

1 n1 = int(input('Informe um número: '))
2 n2 = int(input('Informe outro número: '))
3 p = (n1 > n2)
4 q = (n1 != n2)
5 r = not (p or q) and (not p)
6 print('p =', p)
7 print('q =', q)
8 print('r =', r)

```

Figura 20 – Código com expressões lógicas e relacionais

Fonte: Elaborado pelo Autor.

1.8 Utilizando funções

O Python possui uma biblioteca padrão com funções que podem ser usadas diretamente no código. Além disso, podemos importar diversas outras bibliotecas que podem ser utilizadas para as mais variadas tarefas (BORGES, 2010). A Figura 21 exibe algumas funções da biblioteca padrão. Além da biblioteca padrão, abordaremos funções relacionadas ao tipo textual e também a biblioteca de funções matemáticas.

Função	Funcionamento
abs(n)	Retorna o valor absoluto de n
chr(n)	Retorna o caractere representado pelo número n
ord(c)	Retorna o código correspondente ao caractere c
round(n, d)	Arredonda n considerando d casas decimais
type(o)	retorna o tipo de o

Figura 21 – Algumas funções da biblioteca padrão do Python

Fonte: Elaborado pelo Autor.

1.8.1 Manipulação de dados textuais

Na linguagem Python, o tipo de dado textual (**str**) possui funções relacionadas que auxiliam na manipulação de dados desse tipo (DOWNEY, 2015). Por serem funções associadas ao tipo **str**, é preciso usá-las com o dado desse tipo. Por exemplo, se temos uma variável **x** do tipo **str**, podem chamar uma função **lower()** com a instrução **x.lower()**. A Figura 25 contém alguns exemplos de funções de manipulação de texto. O código da Figura 94 demonstra a utilização de algumas dessas funções.

Função	Funcionamento
s.find(subtexto, ini, fim)	Procura subtexto em s , começando da posição ini até a posição fim . Retorna -1 , se o subtexto não for encontrado.
s.format(x₁, ..., x_n)	Retorna s com os parâmetros x₁, ..., x_n incorporados e formatados.
s.lower()	Retorna o s com todas as letras minúsculas.
s.replace(antigo, novo, n)	Retorna o s substituindo antigo por novo , nas n primeiras ocorrências.

Figura 22 – Algumas funções associadas ao tipo **str**

Fonte: Elaborado pelo Autor.

```

1 nome_completo = input('Informe seu nome completo: ')
2 sobrenome = input('Informe seu sobrenome: ')
3
4 pos = nome_completo.find(sobrenome)
5
6 if pos != -1:
7     print('Seu sobrenome começa na posição ', pos)
8 else:
9     print('Sobrenome não encontrado')
10
11 n = float(input('Informe um número: '))
12 print('{n:.8f}'.format(n=n))

```

Figura 23 – Código com manipulação de dados textuais

Fonte: Elaborado pelo Autor.

1.8.2 Funções matemáticas

Além das funções da biblioteca padrão da linguagem, podemos importar bibliotecas adicionais. Uma dessas bibliotecas é a **math** contendo funções matemáticas (CORRÊA, 2020). A importação de bibliotecas adicionais é realizada com a instrução **import**, normalmente, incluída no início do código. A Figura 24 apresenta algumas das funções disponíveis na biblioteca **math**. Já o código da Figura 25 demonstra como tais funções podem ser utilizadas.

Função	Funcionamento
ceil(x)	Retorna o teto de x
floor(x)	Retorna o piso de x
trunc(x)	Retorna a parte inteira de x
exp(x)	Retorna e^x
log(x, b)	Retorna o logaritmo de x em uma base b . Se a base não for especificada, retorna o logaritmo natural de x
sqrt(x)	Retorna a raiz quadrada de x
pi	Retorna o valor de π

Figura 24 – Algumas funções da biblioteca **math**

Fonte: Elaborado pelo Autor.

```

1 import math
2
3 n = float(input('Informe um número: '))
4 x = n * math.pi
5 print('x = n * pi = ', n)
6 print('Teto de x =', math.ceil(x))
7 print('Piso de x =', math.floor(x))
8 print('Log de x na base 10 =', math.log(x, 10))
9 print('Raiz de x =', math.sqrt(x))

```

Figura 25 – Código com funções matemáticas

Fonte: Elaborado pelo Autor.



Atividade: Resolva os exercícios a seguir. As respostas estão na próxima página, mas é importante que você tente resolver por conta própria e use as respostas apenas para conferência.

1.9 Exercícios

Escreva os códigos em Python para resolver os problemas a seguir:

- A) Calcular a área de um triângulo a partir de sua base e altura.
- B) Escreva um código que receba um número de segundos e converta este número em horas, minutos e segundos. Escreva também um código que faça o contrário.
- C) Considere a fórmula para cálculo de juros simples, $J = (C \times I \times T) / 100$, onde J, C, I e T correspondem a juros, capital, taxa e tempo, respectivamente. Construa um código que solicite ao usuário os valores de C, I e T e calcule J.

As respostas estão na próxima página, mas é importante que você tente resolver os problemas e use as respostas apenas para conferência.

1.10 Respostas dos exercícios

Escreva códigos em Python para resolver os problemas a seguir:

- A) Calcular a área de um triângulo a partir de sua base e altura.

```
1 print('Informe os dados do triângulo')
2 base = float(input('Base: '))
3 altura = float(input('Altura: '))
4 area = base * altura / 2
5 print('A área do triângulo é', area)
```

- B) Escreva um código que receba um número de segundos e converta este número em horas, minutos e segundos. Escreva também um código que faça o contrário.

```
1 print('Segundos para H:M:S')
2 total_segundos = int(input('Informe número total de segundos: '))
3 minutos = total_segundos // 60
4 segundos = total_segundos % 60
5 horas = minutos // 60
6 minutos %= 60
7 print(horas, ':', minutos, ':', segundos)
```

```
1 print('H:M:S para Segundos')
2 print('Informe os dados')
3 horas = int(input('Horas: '))
4 minutos = int(input('Minutos: '))
5 segundos = int(input('Segundos: '))
6 total_segundos = horas * 60 * 60 + minutos * 60 + segundos
7 print('O total de segundos é', total_segundos)
```


C) Considere a fórmula para cálculo de juros simples, $J = (C \times I \times T) / 100$, onde J, C, I e T correspondem a juros, capital, taxa e tempo, respectivamente. Construa um código que solicite ao usuário os valores de C, I e T e calcule J.

```
1 print('Informe os dados para o cálculo dos juros')
2 capital = float(input('Capital: '))
3 taxa = float(input('Taxa de juros: '))
4 tempo = float(input('Tempo: '))
5 juros = capital * taxa * tempo / 100
6 print('O valor dos juros é', juros)
```

1.11 Revisão

Antes de prosseguirmos para a próxima o próximo conteúdo, é importante que você estude, revise o conteúdo e realize pesquisas sobre os conceitos apresentados para ampliar seus conhecimentos.



Mídia digital: Antes de avançarmos nos estudos, vá até a sala virtual e assista ao vídeo “Revisão da Primeira Semana” para recapitular tudo que aprendemos.

Nos encontramos na próxima semana.

Bons estudos!