Raciocínio Computacional

UNIDADE 01

Introdução à programação em Python

Esta unidade tem por objetivo introduzir os conceitos de lógica de programação e algoritmos, bem como os primeiros comandos da linguagem Python, a qual será trabalhada ao longo desta disciplina. Ter o entendimento desses conceitos básicos de forma sólida permitirá um melhor desenvolvimento na sequência da disciplina, possibilitando uma evolução gradual em sua aprendizagem. Seja bem-vindo aos primeiros passos em sua carreira de programador!

Introdução ao raciocínio computacional e linguagem de programação

A tecnologia tem sido algo cada vez mais comum na vida das pessoas, se apresentando no dia a dia de forma tão presente que já se tornou algo natural e imperceptível. Seja pelo uso de um computador, seja de um *smartphone* ou até mesmo do navegador GPS para o carro, o ser humano recebeu um novo título: usuário. Em todos esses aparatos tecnológicos, tem-se, de forma oculta, o trabalho do profissional que desenvolve o *software* que permite a interação entre as pessoas e esses dispositivos: o programador.

O programador tem por função principal fazer com que o usuário se comunique com os dispositivos físicos a partir de um *software*, o qual, também conhecido como aplicativo ou programa, é criado com o encadeamento de comandos padronizados que realizam todo tipo de tarefa. Esses comandos, por sua vez, são transformados em linguagem de máquina para ser interpretados pelos dispositivos e, consequentemente, atender às ordens dadas pelo usuário.

Para realizar essa tarefa, o programador tem à sua disposição vários tipos de linguagem de programação, que podem ser específicos para criar um programa para determinado dispositivo ou genéricos o suficiente para desenvolver funcionalidades para os mais variados equipamentos do mercado. Independentemente da linguagem utilizada, o maior bem que o programador possui é a capacidade de conseguir pensar de forma lógica e estruturada para a criação de todo tipo de programa, para que consiga atuar em qualquer segmento do mercado de programação e/ou adaptar-se a novos desafios sem grandes problemas.

#ParaTodosVerem

©Adobestock

A prática do desenvolvimento de programas é diretamente proporcional à capacidade do profissional de criar soluções lógicas para todo tipo de problema.

#ParaTodosVerem

©Adobestock

Possuir um serrote, um martelo e pregos não faz de qualquer pessoa um marceneiro, uma vez que não é suficiente saber o que cada um desses objetos é ou como eles funcionam.

Para criar um móvel de madeira, deve-se saber quais são os passos a ser empregados e em qual ordem devem ser usadas as ferramentas de marcenaria.

Olhando por esse lado, o programador é um solucionador de problemas, fazendo uso das ferramentas computacionais que possui.

É importante ressaltar que tais ferramentas computacionais são compartilhadas em maior ou menor número entre as linguagens de programação, o que leva àquela frase, de autor desconhecido, comum na área de informática: quem sabe uma linguagem de programação sabe todas. Embora não represente cem porcento da verdade da área, ter domínio do raciocínio lógico e da lógica computacional permite que um programador que conhece determinada linguagem apresente muito menos dificuldades em aprender outras linguagens ou até mesmo outros paradigmas de programação, uma vez que apresenta o principal pré-requisito para tanto: saber resolver problemas de forma estruturada.

Conceito de algoritmo

Um algoritmo é uma sequência de passos padronizados utilizada na solução de determinado problema. É possível tomar como exemplo a execução de uma receita culinária.

#ParaTodosVerem

©Adobestock

Os passos são comuns a várias receitas, como ferver a água, adicionar uma quantidade de gramas de um produto, cozinhar até chegar ao ponto de fervura etc. Contudo, com os mesmos passos é possível fazer uma infinidade de receitas diferentes. Mais importante: se passar para alguém exatamente os mesmos passos e essa pessoa souber como cada um desses passos funciona, é possível reproduzir a mesma receita com a mínima diferença.

Quando esses passos se referem a comandos computacionais, diz-se que é um algoritmo computacional. Os comandos computacionais são padronizados pela lógica de algoritmos e utilizados na solução de problemas também ditos computacionais, ou seja, são utilizados para realizar uma tarefa que envolva a interação *software-hardware*.

Aplicação de algoritmos

Um algoritmo sempre é aplicado na resolução de um problema. O termo “problema” é comum na área de programação para designar uma tarefa que deve ser realizada. Pense, por exemplo, em uma tarefa computacional para calcular a idade de uma pessoa. A resolução do problema é simples: saber o ano de nascimento dela e subtrai-

lo do ano atual. No entanto, em termos de algoritmo computacional, não é tão simples assim, uma vez que, para solucionar a tarefa, devem ser utilizados comandos-padrão de programação.

Um primeiro passo seria mostrar uma mensagem para o usuário na tela pedindo que digite o seu ano de nascimento. Em seguida, armazenar esse valor na memória do computador, para ser utilizado na sequência do algoritmo. Por um comando interno, pegar o ano atual e fazer uma subtração matemática entre esse valor e o valor armazenado na memória do ano de nascimento do usuário, guardando a resposta em outro local da memória. Por fim, mostrar uma mensagem informando a idade do usuário, fazendo uso do valor calculado armazenado.

Mostrar uma mensagem, armazenar um valor na memória e fazer uma subtração de dois valores são todos comandos-padrão na área de algoritmos, os quais devem ser sempre encadeados na sequência correta para resolver um problema.

Tipos de dado primitivo

Com relação ao armazenamento de dados em memória, essa ação faz com que dados obtidos a partir de uma entrada de usuário ou de outros comandos de programação sejam guardados para uso posterior. A memória do computador possui marcações, chamadas endereços, e uma informação, ao ser armazenada, recebe um desses endereços para ser posteriormente acessada.

Dependendo do tipo de informação que está sendo armazenada, ela pode ocupar mais ou menos espaço na memória, bem como permitir que sejam realizados com ela diferentes tipos de operação. É fácil pensar que somar números é algo comum, pois faz parte do contexto matemático, mas e quanto a somar textos? No contexto matemático, isso não existe, mas pode ser possível juntar dois textos para compor uma única frase em um contexto computacional. A essa operação dá-se o nome concatenação.

Em função dessas diferenças em termos de armazenamento e uso, os dados são classificados de acordo com a sua natureza. Em lógica computacional, são quatro os tipos primitivos de dado: inteiro, real, lógico e texto.

Inteiro e **real** são ambos representantes de valores numéricos, porém **inteiro** é utilizado para armazenar valores sem parte fracionária, enquanto **real** é usado para armazenar valores não exatos, que possuem uma ou mais casas decimais. **Lógico** representa um valor dito booleano, que pode ser verdadeiro ou falso, o que permite ter

um dado com apenas dois estados. Por fim, **texto** representa, como o próprio nome diz, um texto qualquer, que não apresenta nenhum dos comportamentos anteriormente descritos.

Tipos primitivos de dado

Tipo	Descrição
Inteiro	Valores sem parte fracionária. Exemplo: idade = 18.
Real	Valores não exatos. Exemplo: PI = 3,14
Lógico	Valor booleano. Exemplo: check = True
Texto	Qualquer texto. Exemplo: mensagem = “Hello, world!”

Esses dados são armazenados em **endereços da memória**, os quais são atribuídos às chamadas **variáveis**, entidades da programação que permitem acessar esses dados e representá-los.

Operadores aritméticos

Os operadores aritméticos são responsáveis por permitir a realização de operações sobre números, de tipo inteiro ou real, seguindo os conceitos matemáticos clássicos. Em lógica de programação, esses operadores são:

Operadores aritméticos

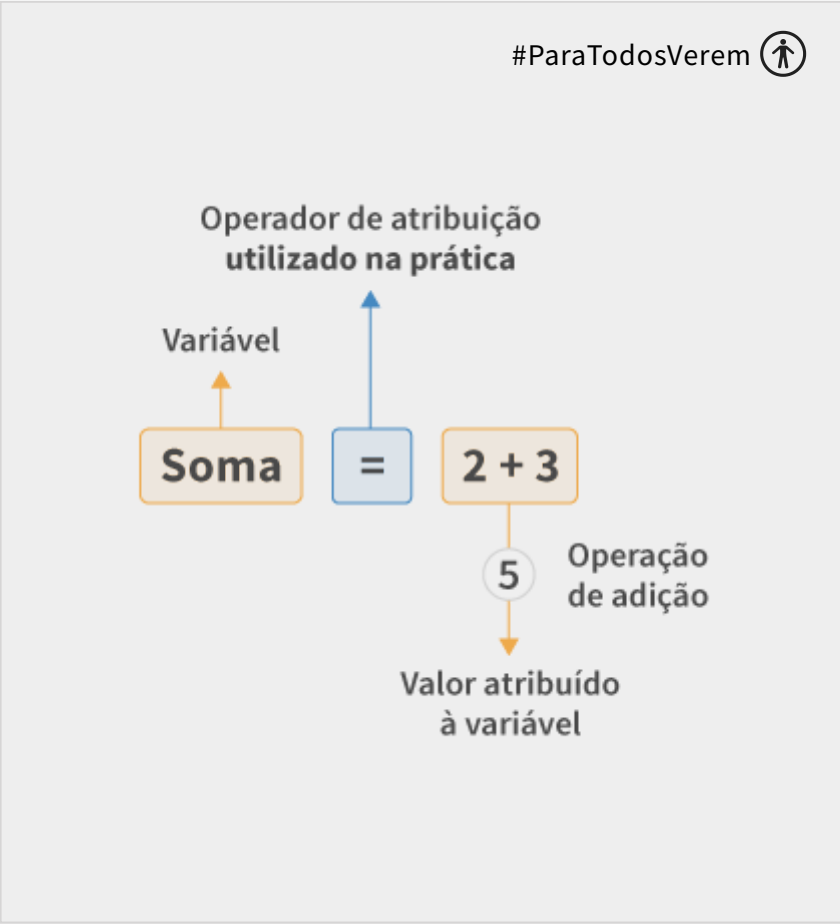
Operação	Operador	Descrição
Adição	+	Realiza a adição matemática de dois números.
Subtração	-	Realiza a subtração matemática de dois números.
Multiplicação	*	Realiza a multiplicação matemática de dois números.

Divisão	/	Realiza a divisão matemática de dois números.
Modulação	%	Retorna o resto da divisão de dois números.

Operador de atribuição

O resultado de uma operação pode ser armazenado em memória para uso posterior. Conforme mencionado anteriormente, um endereço de memória pode ser representado por uma variável. Assim, o ato de atribuir um valor a uma variável significa armazená-lo em um endereço de memória, o qual ficará contido em uma variável, o que, na prática, se trata de atribuir esse valor à própria variável.

Segue um exemplo:



Autor

Em lógica de programação, o operador de atribuição é representado pelo símbolo **<-**, o qual indica que a operação realizada do seu lado direito tem o resultado atribuído à variável que está do seu lado esquerdo. Na prática, as linguagens de programação realizam as atribuições usando o operador de igualdade **=**.

Pseudocódigos

Um algoritmo que realiza determinada tarefa deve ser escrito em pseudocódigo, uma linguagem padronizada para representar a solução que o algoritmo proporciona.

Exemplo de aplicação 1: Elabore um algoritmo que peça ao usuário um número e informe o quadrado dele.

```
1. início
2.     inteiro: num, quad;
3.     escreva (“Qual número deseja elevar ao quadrado?”);
4.     leia (num);
5.     quad <- num * num;
6.     escreva (“O quadrado de”, num, “vale”, quad);
7. fim.
```

EXERCÍCIO

Exercício de fixação 1: Elabore um algoritmo que solicite dois números ao usuário e exiba a soma deles.

Resolução do exercício

Resolução exercício 1

```
início
    inteiro: num1, num2, soma;
    escreva("Digite o primeiro número:");
    leia(num1);
    escreva("Digite o segundo número:");
    leia(num2);
    soma <- num1 + num2;
    escreva("A soma dos números é:", soma);
fim.
```

Exercício de fixação 2: Elabore um algoritmo que solicite ao usuário seu ano de nascimento e calcule sua idade com relação ao ano de 2020, sendo que o usuário já fez aniversário nesse ano.

Resolução do exercício

Resolução exercício 2

```
início
    inteiro: anoNascimento, anoBase, idade;
    anoBase <- 2020;
    escreva("Digite o ano do seu nascimento:");
    leia(anoNascimento);
    idade <- anoBase - anoNascimento;
    escreva("Sua idade é:", idade,"anos");
fim.
```

Exercício de fixação 3: Elabore um algoritmo que solicite ao usuário o nome de uma disciplina e suas quatro notas bimestrais. O algoritmo deve calcular a média dessas notas e exibir uma mensagem informando que a média da disciplina **nome** é **média**.

Resolução do exercício

Resolução exercício 3

```
início
    texto: nomeDisciplina;
    inteiro: nota1, nota2, nota3, nota4, media;
    escreva("Informe o nome da disciplina:");
    leia(nomeDisciplina);
    escreva("Informe a nota do 1º Bimestre:");
    leia(nota1);
    escreva("Informe a nota do 2º Bimestre:");
    leia(nota2);
    escreva("Informe a nota do 3º Bimestre:");
    leia(nota3);
    escreva("Informe a nota do 4º Bimestre:");
    leia(nota4);
    media <- (nota1 + nota2 + nota3 + nota4) / 4;
    escreva("A média da disciplina ",
nomeDisciplina, "é", media);
fim.
```

Exercício de fixação 4: Elabore um algoritmo que solicite o nome de um produto, seu valor e quantidade, informando o valor de compra calculado.

Resolução do exercício

Resolução exercício 4

```
início
    texto: nomeProduto;
    inteiro: quantidadeProduto;
    real: valorProduto, valorCompra;
    escreva("Informe o nome do produto:");
    leia(nomeProduto);
    escreva("Informe o valor do produto:");
    leia(valorProduto);
    escreva("Informe a quantidade do produto:");
    leia(quantidadeProduto);
    valorCompra <- valorProduto *
quantidadeProduto;
    escreva("O valor de compra do produto ",
nomeProduto, "é R$: ", valorCompra);
fim.
```

Exercício de fixação 5: Dê continuidade ao exercício 4 informando que, para pagamento à vista, há 15% de desconto, calculando e exibindo esse valor.

Resolução do exercício

Resolução exercício 5

```
início
    texto: nomeProduto;
    inteiro: quantidadeProduto;
    real: valorProduto, valorCompra, desconto,
valorCompraDesconto;
    desconto <- 0.15;
    escreva("Informe o nome do produto:");
    leia(nomeProduto);
    escreva("Informe o valor do produto:");
    leia(valorProduto);
    escreva("Informe a quantidade do produto:");
```

```
    leia(quantidadeProduto);
    valorCompra <- valorProduto *
quantidadeProduto;
    escreva("O valor de compra do produto ",
nomeProduto, "é R$: ", valorCompra);
    valorCompraDesconto <- valorCompra -
(valorCompra * desconto);
    escreva("O valor de compra do produto ",
nomeProduto, " com desconto a vista é R$: ",
valorCompraDesconto);
fim.
```

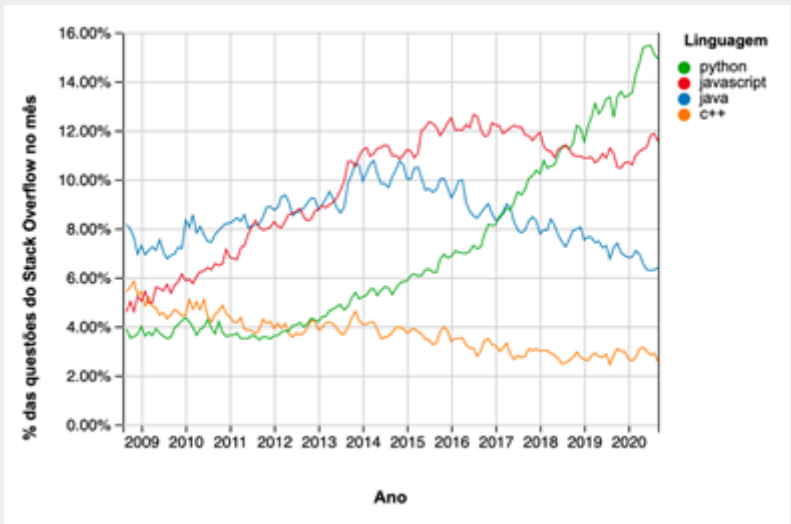
Introdução à linguagem de programação Python

Python é uma linguagem *open source* (código livre), com uma sintaxe simples, porém com grande robustez, contando com recursos equivalentes a outras linguagens comerciais de grande expressão, como C, C++ e Java. Trata-se de uma linguagem interpretada, ou seja, não gera um arquivo executável padrão (.exe, .bat), estando seu interpretador Python e bibliotecas disponíveis para as mais diversas plataformas do mercado, incluindo Windows, macOS e Linux.

O uso de Python vem crescendo exponencialmente nos últimos anos. Tomando por base a plataforma de perguntas e respostas de programação Stack Overflow, é possível verificar esse crescimento frente a linguagens importantes do mercado de programação. O gráfico da Figura 1 mostra as quantidades de perguntas realizadas na plataforma desde 2008, comparando Python com as linguagens C++, JavaScript e Java.

Figura 1. Evolução da linguagem Python na plataforma Stack Overflow

#ParaTodosVerem



Fonte: Stack Overflow (2020)

Stack Overflow é uma plataforma de perguntas e respostas sobre programação, que disponibiliza o serviço Stack Overflow Trends, o qual permite fazer um comparativo da evolução da quantidade de questões efetuadas para determinada linguagem.

Ferramentas para desenvolvimento em Python

Existem várias ferramentas que permitem desenvolver programas em Python, desde aplicações web até Integrated Development Environments (IDEs) completos.

Para o desenvolvimento das atividades desta disciplina, podem ser utilizadas várias ferramentas. Como o código Python é texto, não existe diferença em como elaborar o código nas ferramentas. Entenda que normalmente o professor-tutor irá adotar uma delas, cabendo talvez um questionamento sobre o assunto.

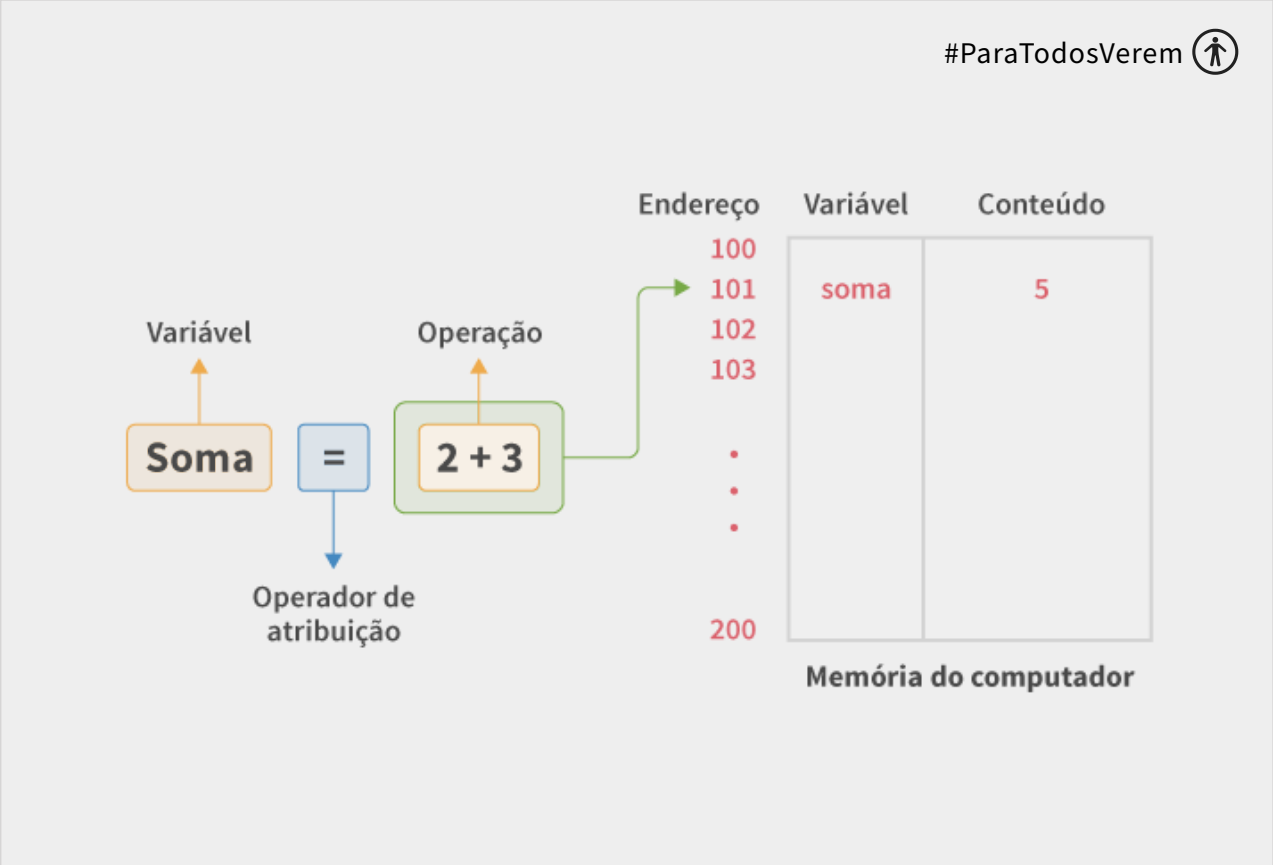
São recomendadas três ferramentas online, caso não deseje instalar na sua máquina, ou não tenha onde instalar: repl.it , online-python, programiz, para exercícios mais simples e pequenas aplicações de testes, mas podem gerar mais trabalho ou dificuldades em exercícios maiores.

Para instalar na sua máquina podem ser usados o IntelliJ, VsCode, ou o Pycharm (mas este tem atualmente um período de 30 dias de teste somente). Estas ferramentas permitem desenvolver diretamente de seu computador, mas em muitas versões exigem também a instalação local do Python.

As ferramentas online tendem a ser mais leves e com menos funcionalidades. As ferramentas instaláveis tendem a ser mais completas contendo diversos recursos que auxiliam no processo de aprendizagem da programação.

Variáveis e constantes

Como mencionado anteriormente, toda vez que se faz necessário armazenar determinado dado na memória do computador, ele é colocado em um endereço, que é atribuído a uma variável, que o armazena e permite acesso ao referido dado.



Autor

Uma **variável** possui um nome que a representa; para criação desse nome, existem algumas regras. O Python aceita a criação de um nome de variável fazendo uso de letras, números e o caractere *underscore* (_), não sendo permitido que o nome de uma variável comece por um número. Existem alguns padrões recomendados para o desenvolvimento em Python, sendo seguido, nesta disciplina, o padrão PEP-0008 ,

segundo o qual é recomendado que o nome de variáveis seja em letra minúscula, com palavras compostas separadas pelo caractere *underscore*. Exemplos de nomes de variáveis em Python segundo o PEP-0008:

```
number
house10
my_name
```

Python é uma linguagem de **tipagem dinâmica** e **forte**. **Tipagem dinâmica** significa que o próprio interpretador do Python determina o tipo de dado de acordo com a informação recebida, sem a necessidade de o programador informá-lo explicitamente. Linguagem de **tipagem estática**, por sua vez, exige que uma variável seja iniciada informando antecipadamente um tipo a partir de uma palavra reservada (exemplos: int, *double*, *float* etc.).

Ser uma linguagem de **tipagem forte** indica que o Python não permite que sejam efetuadas operações com tipos diferentes de dado. Por exemplo:

```
num = 10
name = “Luiza”
print(num + name)
```

Este código vai gerar um erro:

```
TypeError: unsupported operand type(s) for +: 'int' and 'str'
```

Em outras linguagens, como JavaScript, por exemplo, essa operação seria realizada, tendo como saída:

```
10Luiza
```

Este é um exemplo de linguagem de tipagem fraca.

Quando o programador deseja armazenar dados em uma variável que não devem ser mudados ao longo do programa, ele faz uso de uma **constante**, porém constantes em Python são apenas um conceito e uma representação, sendo recomendado pelo PEP-0008 que um dado constante seja representado por uma variável com seu nome em letras maiúsculas, como, por exemplo:

```
NUM = 10
NAME = “Luiza”
```

No entanto, esse tipo de representação não impede que o dado seja modificado ao longo da execução do programa, como ocorre com o uso clássico de constantes em outras linguagens de programação. Assim, o conceito de constantes no Python é apenas representativo.

Entrada e saída de dados

Comunicar-se bem com o usuário é uma parte importante no desenvolvimento de um programa. A forma de comunicação deve ser clara, a fim de obter mais assertivamente as informações necessárias para executar a função desejada na aplicação.

A linguagem Python possui duas funções para comunicação com o usuário: a função *print*, a qual exibe mensagens no console, e a função *input*, que, além de exibir uma mensagem, aguarda que um dado seja digitado. Seguem suas sintaxes:

Sintaxe do comando *print*

```
print(*objects, sep=' ', end='\n', file=sys.stdout,
flush=False)
```

Função *print*

Parâmetro	**Descrição**
objects	Objetos a ser impressos. O * indica que pode ser mais de um.
sep	Separador entre os objetos impressos. Padrão: ' '.
end	Finalizador de linha.
file	Objeto com método *write(string)*. Se omitido, imprime diretamente no console.
flush	*True*/*False*. Força a gravação do dado no destino. Padrão: *False*.

A função *print* não retorna qualquer valor, ou seja, seu retorno é *none*.

Sintaxe do comando *input*

```
input([prompt])
```

Função *input*	
Parâmetro	**Descrição**
prompt (opcional)	Um texto a ser impresso na saída-padrão do sistema, sem pulada de linha.

A função *input* lê uma linha e a converte para uma *string*, removendo caracteres de finalização e, em seguida, retornando essa *string*.

Exemplo de aplicação 2: Elabore um programa que solicite o nome do usuário e mostre uma mensagem de boas-vindas.

```
nome = input("Por favor, digite o seu nome: ")
print("Olá, seja bem-vindo à disciplina", nome)
```

```
C:\
```

```
/Users/user/untitled1/bin/python
/Users/user/projects/untitled1/ExAplic02.py
Por favor, digite o seu nome: Jorge Augusto da Silva
Olá, seja bem-vindo à disciplina Jorge Augusto da Silva


Process finished with exit code 0
```

Exemplo de aplicação 3: Elabore um programa que solicite o nome do usuário e mostre uma mensagem de boas-vindas utilizando como separador "=>" e como finalização três puladas de linha.

```
nome = input("Por favor, digite o seu nome: ")
print("Olá, seja bem-vindo à disciplina", nome, sep='=>',
end='\n\n\n')
```

```
C:\


/Users/user/untitled1/bin/python
/Users/user/projects/untitled1/ExAplic03.py
Por favor, digite o seu nome: Jorge Augusto da Silva
Olá, seja bem-vindo à disciplina=>Jorge Augusto da Silva



Process finished with exit code 0
```

EXERCÍCIO

Exercício de fixação 1: Elabore um programa que solicite ao usuário, separadamente, seu nome e sobrenome e mostre a mensagem: “Seu nome completo: Nome Sobrenome.”, com um espaço na junção dos nomes e um ponto no final, sem pular linha.

Resolução do exercício

Resolução exercício 1

```
nome = input("Por favor, digite o seu nome: ")
sobrenome = input("Por favor, digite o seu
sobrenome: ")
print("Seu nome completo: "+nome+"
"+sobrenome+".")
```

C:\

```
/Users/user/untitled1/bin/python
/Users/user/projects/untitled1/ExFix01.py
Por favor, digite o seu nome: Lucas
Por favor, digite o seu sobrenome:
Oliveira
Seu nome completo: Lucas Oliveira.

Process finished with exit code 0
```

Exercício de fixação 2: Elabore um programa que solicite ao usuário, separadamente, seu nome e sobrenome e mostre a mensagem: “Seu nome é Nome e seu sobrenome é Sobrenome!!!”, com três pontos de exclamação no final da frase, pulando linha.

 Resolução do exercício 

Resolução exercício 2

```
nome = input("Por favor, digite o seu nome: ")
sobrenome = input("Por favor, digite o seu
sobrenome: ")
print("Seu nome é " + nome + " e seu sobrenome é
"+sobrenome+"!!!\n")
```

C:\

```
/Users/user/untitled1/bin/python
/Users/user/projects/untitled1/ExFix01.py
Por favor, digite o seu nome: Lucas
Por favor, digite o seu sobrenome:
Oliveira
Seu nome é Lucas e seu sobrenome é
Oliveira!!!


Process finished with exit code 0
```

Conversão de tipos de dado

Em uma linguagem tipada dinamicamente, nem sempre os dados de entrada correspondem à vontade do programador, pois a tipagem deles depende de como são definidos pelas regras do interpretador Python. Assim sendo, é possível converter tipos de dado para aqueles desejados, a fim de utilizar comportamentos específicos.

Analise o seguinte programa:

```
1. num1 = input("Por favor, digite o primeiro número: ")
2. num2 = input("Por favor, digite o segundo número: ")
3. soma = num1 + num2
4. print("A soma dos números é ", soma)
```

Ao executar esse programa, é exibida a seguinte saída:

C:\

```
/Users/user/untitled1/bin/python
/Users/user/projects/untitled1/Exemplo.py
Por favor, digite o primeiro número: 10
Por favor, digite o segundo número: 20
A soma dos números é 1020


Process finished with exit code 0
```

A princípio, esse comportamento parece estar logicamente incorreto, porém o interpretador Python entende a saída de uma função *input* como uma *string*, sendo a operação de soma entre duas *strings* uma operação válida, porém não no contexto da matemática. Essa operação é chamada concatenação.

Assim, como os dados já foram dinamicamente tipados, a solução para realizar a soma aritmética dos dois números é executar uma conversão de tipos, fazendo com que ambos virem números inteiros.

```
1. num1 = int(input(("Por favor, digite o primeiro número: "))
2. num2 = int(input("Por favor, digite o segundo número: "))
3. soma = num1 + num2
4. print("A soma dos números é ", soma)
```

Ao executar esse programa, é exibida a seguinte saída:

C:\

```
/Users/user/untitled1/bin/python
/Users/user/projects/untitled1/Exemplo.py
Por favor, digite o primeiro número: 10
Por favor, digite o segundo número: 20
A soma dos números é 30


Process finished with exit code 0
```

É possível realizar a conversão de tipos para qualquer tipo de dado no Python. Embora a linguagem tenha todos os seus tipos definidos como objetos, alguns possuem comportamento mais próximo de tipos primitivos, os quais são comumente utilizados em conversão de entradas de dados pelo usuário. São eles:

Funções comuns de conversão de dados simples

Função de conversão	**Descrição**
int()	Converte um dado para um número do tipo inteiro.
float()	Converte um dado para um número do tipo ponto flutuante (com casas decimais).
complex()	Converte um dado para um número complexo.
str()	Converte um dado para sua versão *string*.

É importante ressaltar que uma conversão inválida gera um erro na execução do programa. Por exemplo, ao tentar usar o texto “vinte” no código anterior, tem-se a seguinte saída do programa:

C:\

```
/Users/user/untitled1/bin/python
/Users/user/projects/untitled1/Exemplo.py
Por favor, digite o primeiro número: vinte
Traceback (most recent call last):
  File "/Users/user/projects/untitled1/First.py", line 4,
in <module>
    num1 = int(input("Por favor, digite o primeiro número:
"))
ValueError: invalid literal for int() with base 10:
'vinte'


Process finished with exit code 1
```

Para evitar esse tipo de erro, deve ser efetuado um tratamento de exceção, assunto que será visto nas unidades futuras.

Formatação de saída de dados

O programador, além de solucionar corretamente os problemas computacionais, precisa apresentar a saída de dados de forma clara e organizada.

Existem algumas configurações que permitem que os dados sejam formatados antes de ser exibidos na tela. No exemplo que segue, o resultado da divisão é exibido com todas as casas decimais:

```
num = 6/7
print(num)
```

C:\

```
/Users/user/untitled1/bin/python
/Users/user/projects/untitled1/Exemplo.py
0.8571428571428571


Process finished with exit code 0
```

Python oferece algumas formas de formatar esse tipo de saída. A primeira delas é fazendo uso do operador de modulação (%), que é uma forma mais clássica. Esse tipo de formatação usa um padrão parecido com algumas linguagens clássicas, como C. Seguem alguns exemplos e seus resultados:

Formatação de saída de dados com o operador de modulação

'Num: %f' % num	Num: 0.857143 => padrão de exibição de ponto flutuante com seis casas decimais
'Num: %.2f' % num	Num: 0.86 => formatado com duas casas decimais

'Num: %8.2f' % num	Num: 0.86 => formatado com duas casas decimais, representado com oito caracteres
'Num: %d' % num	Num: 0 => formatado como um valor inteiro
'Num: %s' % num	Num: 0.8571428571428571 => formatado como *string*

O segundo método é pelo uso de *strings* literais formatadas. Também conhecidas como *f-strings*, permitem que sejam colocadas expressões em Python dentro de *strings* no formato {expressão}, prefixando-as com os caracteres "f" ou "F". Seguem exemplos:

Formatação de saída de dados com *f-strings* – *floats* – num = 6/7	
f'Num: {num:f}'	Num: 0.857143 => padrão de exibição de ponto flutuante com seis casas decimais
f'Num: {num:.2f}'	Num: 0.86 => formatado com duas casas decimais
f'Num: {num:8.2f}'	Num: 0.86 => formatado com duas casas decimais, representado com oito caracteres
Formatação de saída de dados com *f-strings* – inteiros – num = 10	
f'Num: {num:8d}'	Num: 10 => formatado com, no mínimo, oito caracteres (à esquerda)
Formatação de saída de dados com *f-strings* – *strings* – t1 = "Hello", t2 = "World"	
f'{t1:10}{t2:10}'	Hello World => formatado com, no mínimo, dez caracteres, sem alinhamento
f'{t1:^10}{t2:^10}'	Hello World => formatado com, no mínimo, dez caracteres, centralizado

f'{t1:>10}{t2:>10}'	Hello World => formatado com, no mínimo, dez caracteres, à direita

O terceiro método – um dos mais utilizados – faz uso do método *format*.

Formatação de saída de dados com o método *format* – *floats* – num = 6/7

'Num: {0:f}'.format(num)	Num: 0.857143
'Num: {0:.2f}'.format(num)	Num: 0.86
'Num: {0:8.2f}'.format(num)	Num: 0.86

Formatação de saída de dados com o método *format* – inteiros – num = 10

'Num: {0:8d}'.format(num)	Num: 10

Formatação de saída de dados com o método *format* – *strings* – t1 = "Hello", t2 = "World"

'{0:10} {1:10}'.format(t1, t2)	Hello World
'{0:^10} {1:^10}'.format(t1, t2)	Hello World
'{0:>10} {1:>10}'.format(t1	Hello World

, t2)

Exemplo de aplicação 4: Solicite ao usuário dois produtos, com suas respectivas quantidades e preços, e mostre esses dados formatados como tabelas.

```
1. prod1 = input("Digite o nome do primeiro produto: ")
2. quant1 = int(input("Digite a quantidade do primeiro
   produto: "))
3. valor1 = float(input("Digite o valor do primeiro produto:
   "))
4. prod2 = input("Digite o nome do segundo produto: ")
5. quant2 = int(input("Digite a quantidade do segundo produto:
   "))
6. valor2 = float(input("Digite o valor do segundo produto:
   "))
7. print(f'{"Produto":20}|{"Quantidade":^12}|{"Valor":>10}')
8. print(f'{prod1:20}|{quant1:^12}|{valor1:10.2f}')
9. print(f'{prod2:20}|{quant2:^12}|{valor2:10.2f}')
```

C:\

```
/Users/user/untitled1/bin/python
/Users/user/projects/untitled1/ExAplic04.py
Digite o nome do primeiro produto: Caneta
Digite a quantidade do primeiro produto: 20
Digite o valor do primeiro produto: 7.5
Digite o nome do segundo produto: Mochila
Digite a quantidade do segundo produto: 3
Digite o valor do segundo produto: 117.30
Produto              | Quantidade |     Valor
Caneta               |     20     |      7.50
Mochila              |     3      |    117.30


Process finished with exit code 0
```

Comentários

Comentário, em qualquer linguagem de programação, é um recurso importante para organização e documentação de códigos. Existem dois tipos de comentário em Python: **comentário *inline*** e **comentário *multiline***.

O **comentário *inline*** é aplicado em uma única linha, sendo demarcado pelo caractere cerquilha (#). Tudo que vier antes da linha será desconsiderado pelo interpretador Python. Por exemplo:

```
# Tudo que vem antes desta linha é desconsiderado pelo
interpretador Python
print("Teste") # O mesmo ocorre na sequência desta linha
```

O **comentário *multiline*** é identificado por três apóstrofos seguidos (''') no início e no fim do comentário, o que delimita um bloco. Por exemplo:

```
'''
Este arquivo Python resolve um problema computacional.
Arquivo criado em 18/10/2020.
Última alteração: 28/10/2020.
'''
```

Quanto melhor comentado for um código, mais fácil será o entendimento futuro pelo programador que o criou, bem como por outro profissional que venha a manter o código.

Placeholder e *format*

Neste vídeo de demonstração de código, vermos como utilizar *placeholder* e função *format* para formatação da saída de dados no console em Python.

Formatação da saída de dados (placeholder e format)

Conclusão

A respeito dos conceitos introdutórios ao raciocínio computacional e à linguagem de programação Python, podemos concluir que:

- A resolução de problemas computacionais deve seguir passos padronizados.
- Deve-se utilizar uma linguagem comum, que possa ser reproduzida por quem a conheça e tenha acesso aos mesmos passos, na forma de um algoritmo.
- A lógica de programação é conceito-chave para aprender qualquer linguagem de programação.
- A linguagem Python vem crescendo exponencialmente nos últimos anos, sendo uma ótima primeira opção de aprendizagem na carreira de programador.
- Python é uma linguagem de tipagem dinâmica e forte.
- Para desenvolvimento em Python, temos várias ferramentas, sendo uma das mais completas o PyCharm, da empresa JetBrains.
- Em programação, podemos guardar informações para uso futuro utilizando variáveis.
- As variáveis representam endereços de memória onde estão armazenados dados de um tipo identificado pelo interpretador Python, que, porém, pode ser convertido para outros tipos.
- Entradas e saídas de informação podem ser efetuadas com o uso das funções *print* e *input* da linguagem.

- As saídas podem ser formatadas a partir de três métodos: operador de modulação, *f-strings* e método *format*.
- Comentários no código são importantes para o entendimento futuro daquilo que foi criado no presente.
- Em Python, podem ser usados comentários *inline* e *multiline*.

Referências

BANIN, S. L. **Python 3**: conceitos e aplicações uma abordagem didática. São Paulo: Érica, 2018.

FORBELLONE, A. L. V.; EBERSPÄCHER, H. F. **Lógica de programação**: a construção de algoritmos e estruturas de dados. São Paulo: Prentice Hall, 2005.