# SUMÁRIO

[**SUMÁRIO 1**](#_11agubohmyoa)

[**Programação Orientada a Objetos 2**](#_t4yjix6t6a0)

[**Apresentação da disciplina e preparação do ambiente de desenvolvimento 2**](#_hzps4ktscmq4)

[Configuração do ambiente de desenvolvimento 3](#_aidqtei601f)

[Princípios norteadores e um guia de estilo para Python 4](#_2ljwcxbj03m2)

[**Revisão de Python básico 5**](#_3d06m9q1ytrk)

[Tipos de dados 6](#_n65hm0zdto4i)

[int 6](#_ciw0o2qkkzvl)

[float 6](#_vc06p2fuok5v)

[bool 7](#_qq70m07kql76)

[str 7](#_daxp3u2jxy8g)

[Variáveis 9](#_tarrhaoo14j0)

[Operadores 10](#_t7a605coxo94)

[Funções 11](#_b5owgbu5yj5j)

# Programação Orientada a Objetos

Esta disciplina irá aprofundar o estudo da linguagem Python e apresentar os conceitos de Programação Orientada a Objetos (POO), tanto a teoria, que pode ser aplicada também em diversas outras linguagens de programação, quanto a sua aplicação em Python, habilitando-nos dessa forma a resolver problemas cada vez mais complexos.

O conteúdo visto na disciplina de Linguagem de Programação será essencial para a continuação dos estudos. Todos os conceitos e estruturas aprendidos lá serão utilizados ao longo desta disciplina.

A disciplina está organizada da seguinte forma: começando com uma revisão de Python básico, a fim de retomar os conceitos vistos em Linguagem de Programação, antes de seguir para a Programação Orientada a Objetos. Serão apresentados os pilares de POO e como eles são aplicados em Python, o que é e como se construir uma classe, criar objetos e fazer com que eles interajam entre si, através de seus métodos.

Essa disciplina também inclui tópicos como a manipulação de arquivos com Python, a criação e utilização de ambientes virtuais, a modularização e criação de pacotes, o gerenciamento de dependências e a utilização de testes automatizados, além de uma introdução aos princípios do SOLID e aos Padrões de Projeto. Esses recursos são essenciais para o bom desenvolvimento de software e irão aprimorar suas habilidades como desenvolvedor e contribuir para sua formação.

Disciplina elaborada e apresentada pelo professor Rafael Maximo Carreira Ribeiro.

# Apresentação da disciplina e preparação do ambiente de desenvolvimento

A disciplina de Programação Orientada a Objetos é uma continuação do que foi visto em Linguagem de Programação (LP). Nesta disciplina vocês conhecerão mais estruturas de programação utilizadas em Python, e que podem também ser aplicadas em outras linguagens de programação, para poder resolver problemas ainda mais complexos.

Tudo que você aprendeu em LP será aproveitado aqui: variáveis, tipos de dados, estruturas de decisão, estruturas de repetição, listas, tuplas e funções. Isso é na realidade a base para todas as disciplinas que envolvem programação no curso e não será diferente para esta. Utilizaremos tais conceitos para aprender um novo paradigma de desenvolvimento de software, no qual abordamos a representação dos problemas baseada na construção de objetos e na interação entre eles, traçando um paralelo mais próximo entre a realidade e a programação.

Esse paradigma é conhecido como Programação Orientada a Objetos, que chamaremos de POO de agora em diante. Aprenderemos o que são objetos, classes, métodos e atributos, veremos qual a relação entre eles e conceitos como abstração, encapsulamento, herança e polimorfismo, que formam os pilares de POO.

Além de POO, aprenderemos também sobre como criar módulos em Python, estudaremos como podemos testar nossas aplicações de maneira automatizada e veremos uma introdução aos princípios do SOLID e aos padrões de projeto. Estes fatores contribuem para um código de manutenção mais fácil, com menos erros ou bugs e portanto com maior qualidade.

Esta disciplina é fundamental para sua formação na área de tecnologia da informação, mesmo que você não venha a trabalhar diretamente como desenvolvedor, os conceitos aprendidos aqui com certeza serão um diferencial na sua carreira.

## Configuração do ambiente de desenvolvimento

A disciplina de Programação Orientada a Objetos utiliza como base a linguagem de programação Python, dando sequência ao que foi visto em LP. Você precisará ter instalado em seu computador a versão 3 do Python, de preferência a mais recente, mas para esta disciplina qualquer versão igual ou superior a 3.7 será suficiente. Se precisar, baixe a versão mais recente em <https://www.python.org/downloads/>. Este é o mesmo instalador utilizado na disciplina de LP, e a configuração de instalação será exatamente a mesma. O que muda agora é o editor que utilizaremos para escrever os programas.

Um programa de computador ou um projeto pode ser desenvolvido utilizando apenas o bloco de notas. Um arquivo de código-fonte em Python nada mais é que um arquivo de texto codificado utilizando a tabela UTF-8 e com a extensão \*.py ao invés de \*.txt. O programa pode ter sido feito no bloco de notas, e podemos abrir um prompt de comando no windows e executá-lo diretamente, o único requisito é possuir o interpretador do Python instalado e acessível.

A linguagem Python possui o próprio Ambiente de Desenvolvimento Integrado ou IDE, da sigla em inglês Integrated Development Environment, chamado IDLE. A IDLE é uma IDE voltada para o aprendizado da linguagem Python, o L vem de Learning, e é instalado automaticamente junto com a instalação do interpretador do Python (nos sistemas Windows, para Linux pode ser necessária a instalação à parte da IDLE).

A IDLE é um programa que contém um console interativo para executar comandos do Python, chamado Shell, e um editor de texto próprio, com realce de sintaxe e outras ferramentas simples.

Uma IDE, de forma geral, é um software que provê facilidade quando estamos escrevendo um programa, com o intuito de aumentar a produtividade ao integrar diversas ferramentas em um mesmo ambiente. Como vimos, a IDLE provê a coloração do código (realce de sintaxe), que facilita a identificação de comandos e estruturas, e a integração com a Shell, que nos permite executar o programa escrito e inspecionar as variáveis criadas após sua execução.

A definição do que um programa precisa para ser considerado uma IDE pode variar, mas deve incluir pelo menos:

* Um editor de texto para a linguagem em questão;
* Possibilidade de gerenciamento de arquivos;
* Construção automática, isto é, um compilador ou interpretador capaz de gerar o código binário e realizar os processos necessários para que o código possa ser executado;
* Uma ferramenta de depuração (debugger).

Outras funcionalidades muito comuns em IDEs modernas incluem:

* Função de autocompletar os comandos digitados;
* Integração com ferramentas para, entre outros:
* Versionamento (GIT);
* Construção de interfaces gráficas (GUI); e
* Construção de diagramas.

Utilizaremos o Visual Studio Code, que abreviaremos para VSCode, uma IDE moderna desenvolvida pela Microsoft, de código aberto, com suporte a diversas linguagens e totalmente configurável. O VSCode (<https://code.visualstudio.com/>) está disponível para Windows, Linux e MacOS, vem com o básico para começarmos a programar e permite a instalação de extensões disponibilizadas pela comunidade para atender uma grande variedade de necessidades.

O VSCode é uma IDE de propósito geral, ou seja, pode ser utilizado para diversas linguagens simultaneamente. Ele provê grande parte das facilidades mencionadas, além de ser leve e ter uma ótima integração com a linguagem Python. No entanto, a instalação padrão vem somente com as funcionalidades básicas, e as ferramentas específicas para cada linguagem devem ser instaladas como extensões, usando a própria interface do VSCode. Para esta disciplina será utilizada a extensão do Python, desenvolvida pela própria Microsoft.

## Princípios norteadores e um guia de estilo para Python

Python é uma linguagem de código aberto, e como tal, alterações e melhorias na linguagem são propostas[[1]](#footnote-0), discutidas, avaliadas e implementadas pela própria comunidade de desenvolvedores. Para gerenciar todo esse processo, existe a Python Software Foundation, organização sem fins lucrativos responsável por manter e avançar o desenvolvimento da linguagem, veja mais em https://www.python.org/psf-landing/.

Uma das principais ideias que guiaram o criador da linguagem Python, Guido van Rossum, é que programas são lidos com muito mais frequência do que são escritos (Python Software Foundation, 2021).

Pense em uma aplicação qualquer, um trecho de código é escrito apenas uma vez, mas conforme a aplicação evolui, eventualmente precisaremos revisitá-lo, seja para corrigir um bug, para aprimorar o algoritmo utilizado ou para adicionar uma nova funcionalidade às já existentes, e para fazer isso corretamente, é imprescindível entender o que o código que já está lá faz, antes de podermos incluir as modificações necessárias.

Agora imagine que você precise editar um código escrito por outro desenvolvedor, que há anos não trabalha mais na empresa, e que já teve alterações feitas por diversas outras pessoas. Se cada um programar da forma que bem entender, muito rapidamente a situação fica insustentável e você gastaria muito mais tempo apenas tentando entender o que já está ali do que de fato implementando algo novo.

Por isso podemos dizer que a legibilidade e consistência do código são extremamente importantes, tão importantes quanto o algoritmo em si. Para isso, temos um guia de estilo desenvolvido pela comunidade Python e apresentado na proposta de melhoria número 8, a PEP 8 (Rossum, G. V., Warsaw, B., Coghlan, N., 2013).

Tim Peters (1999), resumiu os princípios que guiaram o desenvolvimento da linguagem Python, totalizando 19 aforismas, em um poema que leva o nome de “Zen do Python”. Estes princípios foram incluídos oficialmente na PEP 20, em 2004 (Peter, T., 2004) e podem ser acessados na Shell do Python, com o seguinte comando:

Code:

>>> import this

PEP 8 - Guia de Estilo

O guia de estilo para o Python é bastante detalhado e foi desenvolvido pensando justamente em aprimorar tanto a legibilidade quanto a consistência do código. Ele cobre os principais pontos, de como nomear variáveis e funções a como quebrar linhas que ficam muito compridas, passando por quando é recomendado pular linhas no código e como devemos usar variáveis booleanas em estruturas condicionais, entre muitas outras recomendações.

O guia pode ser lido na íntegra em <https://www.python.org/dev/peps/pep-0008/>, mas não é algo que deve ser lido uma vez e decorado. Pelo contrário, deve ser incorporado organicamente como uma filosofia de desenvolvimento, sendo consultado sempre que necessário.

Como todo guia ou recomendação, isso não diz respeito às regras sintáticas do Python, portanto o fato de um trecho de código não segui-lo não surtirá nenhum efeito do ponto de vista de execução desse código, mas pode reduzir bastante sua legibilidade por outras pessoas que precisarem editá-lo (incluindo você mesmo no futuro).

Conforme avançamos na disciplina, falaremos um pouco mais sobre uma ou outra recomendação, de modo que vocês possam aprendê-las aos poucos, aplicando-as de maneira orgânica em seus códigos de acordo com a necessidade. Outra coisa muito importante quando se trata de um guia de estilo geral como este, é saber quando não se deve usá-lo. Não existe uma resposta correta para essa questão, portanto sempre que estiver em dúvida, não hesite em perguntar e discutir com colegas a melhor abordagem (use o “Zen do Python” para guiar a discussão).

Como dito anteriormente, o guia de estilo foi pensado para melhorar a consistência, mas não adianta ser consistente com o guia se isso faz o código inconsistente com outra parte do código. É comum projetos ou empresas terem o seu próprio guia de estilo, que pode ser ou não baseado no guia geral, portanto é importante saber quando seguir e quando ignorar uma recomendação.

*Um guia de estilo fala sobre consistência. Consistência com este [PEP 8] guia é importante. Consistência interna em um projeto é mais importante. Consistência interna de um módulo ou função é ainda mais importante. (Rossum, G. V., Warsaw, B., Coghlan, N., 2013)*

A PEP8 lista também algumas razões e situações em que o mais sensato é ignorar a recomendação, como por exemplo:

1. Se por algum motivo aplicar a recomendação torna o código menos legível, incluindo para alguém já familiarizado com o guia da PEP8;
2. Para manter a consistência com o código ao redor, que também quebra determinada recomendação (talvez por motivos históricos, embora esta possa ser uma boa oportunidade de “limpar” a bagunça de outra pessoa);
3. Porque o código em questão foi escrito antes do lançamento da recomendação e não há nenhuma outra razão para modificá-lo;
4. Quando o código em questão precisa ser compatível com uma versão mais antiga do Python que não suporta o recurso utilizado pela recomendação.

Para saber mais a respeito do guia de estilo do Python, acesse How to Write Beautiful Python Code With PEP 8 – Real Python - <https://realpython.com/python-pep8/>.

# Revisão de Python básico

Como dito anteriormente, a Programação Orientada a Objetos é construída em cima dos conceitos visto em Linguagem de Programação, e portanto iremos começar com uma revisão geral de Python básico, vendo neste capítulo os seguintes conceitos:

* Tipos de dados: int, float, bool, str;
* Variáveis;
* Operadores; e
* Funções.

## Tipos de dados

Os tipos básicos de dados que veremos neste capítulo são o números inteiros, números em ponto flutuante (representando os números reais), valores booleanos e sequências de caracteres (texto). A descrição completa dos tipos integrados à linguagem pode ser acessada em PSF (2021c).

Para saber o tipo de um valor em Python podemos usar a função integrada type na Shell do Python:

code:

>>> type(3)

<class 'int'>

>>> type("3")

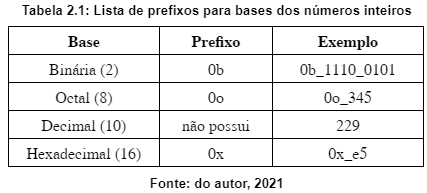
<class 'str'>

Nesta disciplina vamos aprender em breve sobre o que são classes em POO, e veremos que toda classe que criamos automaticamente define um tipo de objeto, por isso a função type retorna essa representação dizendo qual é a “classe” do objeto que passamos para ela como argumento.

### int

O tipo int representa os números inteiros. Em Python não há um limite para o tamanho máximo de um número inteiro, estando apenas limitado pela quantidade de memória disponível para sua alocação.

Os números inteiros podem ser representados nas bases binária, octal e hexadecimal, sendo nesse caso escritos com um prefixo indicativo da base, como mostra a Tabela 2.1. Para a base decimal não há necessidade de nenhum prefixo e é a base adotada por padrão, e podemos separar os dígitos por um sinal de sublinhado para facilitar sua leitura (PSF, 2021a).



Note que o número é sempre o mesmo (faça o teste digitando os exemplos na Shell), e será guardado na memória física sempre em binário, pois essa é a única linguagem que nossos computadores tradicionais entendem, a única coisa que muda é a sua representação. Isso é análogo a medir a distância entre duas cidades em milhas ou quilômetros, o valor representado muda de acordo com a unidade que escolhemos, mas a distância real entre as cidades é sempre a mesma.

### float

O tipo float representa os números reais e ao contrário do tipo int, não é ilimitado. Sua precisão e os limites de variação permitidos aos valores variam de acordo com a implementação do interpretador do Python. Isso ocorre devido a representação de um número que em binário não é exato, da mesma forma que ⅓ não é exato em decimal. Essa impossibilidade de representação exata pode levar a problemas de aproximação ou representação dos quais precisamos estar cientes, pois não é um “bug” do Python, mas sim um problema comum da computação. Esse assunto é tratado em maior detalhe em PSF (2021b).

Veja alguns exemplos de números representados em ponto flutuante no Python:

3.14 10. .001 1e10 5.3e-4

Lembrando que os dois últimos exemplos são números representados em notação científica.

### bool

Há apenas dois objetos constantes que são do tipo bool, e representam os valores de verdadeiro e falso, escritos, respectivamente, como True e False (PSF, 2021d). Observe que não há aspas, pois não são strings.

A função integrada bool() pode ser usada para converter valores em Python para um booleano. Para entender melhor como funciona a conversão de valores para booleano, confira a documentação do Python (PSF, 2021c).

### str

O tipo str é a abreviação do termo string, que representa dados de texto no Python. A partir da versão 3 do Python, toda string segue a codificação Unicode.

Para definir uma string podemos usar tanto aspas simples quanto aspas duplas, com o mesmo resultado:

code:

>>> 'Olá mundo!'

'Olá mundo!'

>>> "Olá mundo!"

'Olá mundo!'

Podemos também construir strings com mais de uma linha utilizando três aspas seguidas (tanto simples quanto duplas):

code:

>>> texto = '''Olá mundo!

... escrito em

... várias linhas'''

>>> texto

Olá mundo!\nescrito em\nvárias linhas

>>> print(texto)

Olá mundo!

escrito em

várias linhas

Observe que as quebras de linha são mantidas e representadas pelo caractere \n. Dizemos que a contra barra \ é usada para escapar a letra n, dando a ela um significado especial, que representa uma quebra de linha.

Ao compararmos duas strings, elas serão iguais apenas se possuírem exatamente os mesmos caracteres e na mesma ordem. Quando comparadas com os operadores de maior e menor, o Python faz a comparação colocando-as em ordem alfabética, ou seja, não importa o tamanho das strings, mas sim qual delas tem letras menores no alfabeto, sendo a posição no alfabeto definida com base no código Unicode do caractere. Para aprender mais veja a documentação do Python (PSF, 2021g).

Podemos concatenar strings, usando o operador + e também criar strings formatadas de diferentes maneiras. O método recomendado para formatação de strings em novos projetos (Bader, 2018) é chamado de “strings literais formatadas” ou f-strings, e consiste em uma string comum prefixada com a letra f ou F (antes da abertura das aspas) e com os valores ou expressões que serão formatados inseridos entre pares de chaves.

Essa forma de formatação é compatível apenas com versões do Python superiores à 3.6 (PSF, 2021e).

Veja o seguinte exemplo:

code:

>>> texto = f'4 + 7 = {4 + 7}, certo?'

>>> texto

'4 + 7 = 11, entendeu?'

Note que a expressão entre o par de chaves é avaliada e o resultado inserido no mesmo ponto em que está escrita na string. A expressão é avaliada em tempo de execução, o que permite o uso de f-strings com variáveis e expressões, como no próximo exemplo:

code:

>>> salario = 3500.0

>>> f'20% a mais em R$ {salario} dará R$ {salario\*1.2}'

'20% a mais em R$ 3500.0 dará R$ 4200.0'

No exemplo acima, seria interessante exibir os valores monetários com duas casas decimais. Para especificar como queremos que um valor seja formatado em uma f-string, adicionamos : após o valor e complementamos com alguns especificadores.

Para formatar um dado do tipo float, podemos usar o seguinte padrão de formatação:

f'{<valor>:<colunas>.<decimais>f}'. Onde:

* colunas: é a quantidade mínima de colunas reservadas para o valor na string formatada, note que cada caractere do valor ocupa uma coluna, inclusive o ponto. Pode ser omitido;
* decimais: é o número total de casas decimais que serão representadas na string, com o valor sendo arredondado caso necessário;
* f: indica que a formatação será feita para o tipo float, podendo haver uma conversão entre tipos compatíveis, como o int.

Veja o código abaixo e, em seguida, tente reescrever o exemplo inicial do salário, porém formatando os valores em reais com duas casas decimais.

code:

>>> pi = 3.14159265

>>> f'{pi:f}' # sem especificar, o padrão é de 6 casas decimais

'3.141593'

>>> f'{pi:.3f}' # note o arredondamento na última casa decimal

'3.142'

>>> f'{pi:7.3f}' # 7 colunas totais, 3 casas decimais

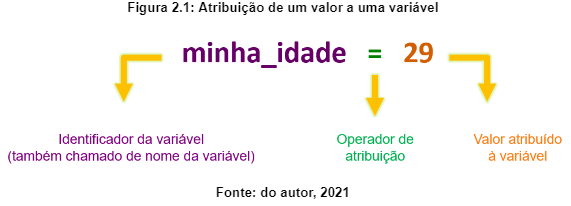
' 3.142'

Existem outros especificadores para formatação de float, como exibição em notação científica, e também para outros tipos, como int e string, além de opções para alterar o alinhamento do texto e o caractere padrão de preenchimento dos espaços vazios. A lista completa com as explicações e exemplos de uso pode ser vista na documentação oficial do Python (PSF, 2021f).

## Variáveis

Uma variável é um espaço de memória associado a um identificador, ou seja, um nome, e serve para guardar valores que o programa poderá acessar e modificar. Toda variável possui um identificador único, que poderá ser referenciado no código sem ambiguidade.

Em Python, uma variável é criada no momento em que um valor é atribuído a um identificador válido. A atribuição é feita através do operador de atribuição, o sinal de igual, colocando-se um identificador à esquerda e um valor à direita deste operador, conforme a Figura 2.1.



Assim como a nomenclatura evidencia, o conteúdo de uma variável pode “variar”, ou seja, uma mesma variável pode guardar valores diferentes em momentos diferentes de um programa em Python. Lembre-se: uma variável só pode guardar um valor por vez, portanto a cada nova atribuição o valor atual será sobrescrito pelo novo.

code:

>>> a = True

>>> type(a)

>>> a = 123

>>> a = 'linda casa amarela'

>>> a = 4.40

>>> a

>>> type(a)

Um destaque importante da linguagem Python é que o tipo do dado está relacionado ao valor atribuído e não a variável que recebeu esse valor, diferentemente de outras linguagens de programação como C, C++ e Java, para citar alguns exemplos.

Um nome ou identificador de uma variável é formado por uma sequência de um ou mais caracteres, de acordo com as seguintes regras:

* Pode conter apenas letras, números e o símbolo de sublinhado (nenhum outro caractere especial é aceito);
* Não pode começar com um número;
* Não pode ser uma palavra reservada.

Ao criar uma variável, recomenda-se utilizar identificadores que sejam concisos, porém descritivos:

* idade é melhor que i;
* tamanho\_nome é melhor que tamanho\_do\_nome\_da\_pessoa.

No entanto, evite abreviar nomes, escrevendo-os por extenso para melhorar a legibilidade do código, por exemplo:

* sobrenome é melhor que sbrnome;
* litros é melhor que ltrs;
* data\_criacao é melhor que dt\_cri.

A PEP 8 recomenda usarmos apenas letras minúsculas e sem acentuação para criar identificadores de variáveis e funções, separando as palavras com um símbolo de sublinhado para melhorar a legibilidade.

Vale lembrar que o Python é uma linguagem case-sensitive, diferenciando letras maiúsculas de minúsculas, portanto é preciso prestar atenção na grafia exata dos identificadores, pois meu\_nome não é o mesmo que Meu\_Nome ou MEU\_NOME.

O Python possui um conjunto de palavras reservadas, chamadas em inglês de keywords (palavras-chave). Essas palavras não podem ser usadas como identificadores, pois possuem um papel especial para o interpretador. O Python 3.9.1 possui 36 palavras reservadas, porém esse número pode variar entre versões diferentes, para saber quais são as da versão que está usando, execute a seguinte instrução na Shell do Python:

code:

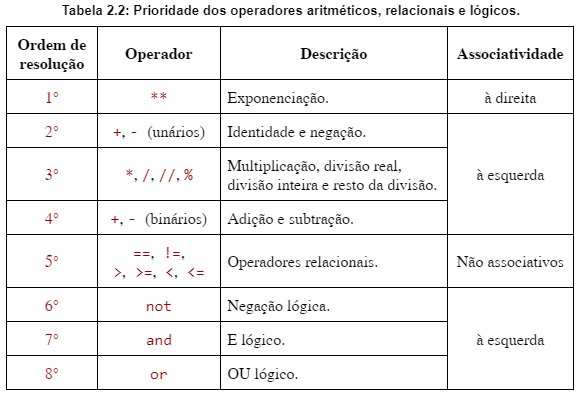
>>> help('keywords')

## Operadores

Vamos relembrar três tipos de operadores aqui:

* Aritméticos: usados para realizar operações matemáticas entre os operandos;
* Relacionais: usados para comparar dois objetos em Python;
* Lógicos: usados para realizar operações lógicas com valores booleanos.

A Tabela 2.2 mostra a ordem em que os operadores são resolvidos pelo Python, de acordo com sua prioridade, sendo o operador de maior prioridade resolvido primeiro.



## Funções

Para criarmos uma nova função em Python, utilizamos a palavra chave def seguida do nome da função e um par de parênteses, dentro dos quais listamos os eventuais parâmetros. Em seguida colocamos : para indicar o fim do cabeçalho[[2]](#footnote-1) da função e início de seu bloco de código, que deve começar na linha seguinte com indentação de 4 espaços em relação à coluna inicial do cabeçalho.

code:

def <nome da função>([<parâmetros>]):

<bloco de código da função>

Parei - pág. 9

1. Esse processo é realizado através das PEP’s, ou Propostas de Melhoria do Python (do inglês Python Enhancement Proposal). Veja mais em https://www.python.org/dev/peps/. [↑](#footnote-ref-0)
2. Em Python, o nome da função mais a quantidade e ordem dos seus parâmetros é também chamado de assinatura da função. Em outras linguagens, esse conceito pode incluir também o tipo dos parâmetros e o tipo do retorno da função, mas tudo isso depende de como a linguagem trata a tipagem de dados. [↑](#footnote-ref-1)