Revista Brasileira de Estudos de Gestão e Desenvolvimento Regional
v. 09, n. 03, p. 138-145, 2026
Seção de Artigos

PRINCIPAIS DIFERENÇAS ENTRE A LINGUAGEM C E A LINGUAGEM *PYTHON*

MAIN DIFFERENCES BETWEEN THE C AND PYTHON LANGUAGES

PRINCIPALES DIFERENCIAS ENTRE LOS LENGUAJES C Y PYTHON

João Gabriel de Jesus Pires Quintanilha
Universidade do Estado de Mato Grosso - UNEMAT
e-mail: jgjquintanilha@gmail.com

Dra. Janecler Foppa
https://orcid.org/0000-0002-8906-4195
Universidade do Estado de Mato Grosso - UNEMAT
e-mail: janecler.foppa@unemat.br

Gabriel de Carvalho Donel
Universidade do Estado de Mato Grosso - UNEMAT
e-mail: gabriel.donel@unemat.br

Pirakta Tawa Ikpeng
Universidade do Estado de Mato Grosso - UNEMAT
e-mail: piraktaikpeng@gmail.com

Submissão em: 18/05/2026 Aceito em: 12/06/2026

RESUMO
Este artigo relata as principais diferenças das linguagens C e *Python* marcantes em filosofia, estrutura e aplicação. A metodologia utilizada foi de natureza qualitativa e abordagem bibliográfica. A linguagem C é compilada, procedural e de tipagem estática, oferecendo alto desempenho e controle direto da memória por meio de ponteiros. Já *Python* é interpretada, orientada a objetos e de tipagem dinâmica, priorizando simplicidade, legibilidade e rapidez no desenvolvimento. Enquanto C exige maior conhecimento técnico e sintaxe mais detalhada, *Python* apresenta código mais enxuto e fácil de aprender. C é muito utilizada em sistemas de baixo nível e aplicações que exigem alta performance. *Python* se destaca em áreas como automação, ciência de dados e desenvolvimento rápido de aplicações. Outra diferença importante está no gerenciamento de memória: C exige controle manual, enquanto *Python* utiliza *Garbage Collector* automático. Assim, a escolha entre as duas linguagens depende das necessidades do projeto, equilibrando desempenho, simplicidade e produtividade.
Palavras-chave: Linguagem compliada, Linguagem interpretada, Desempenho, Simplicidade

ABSTRACT
This article reports on the main differences between the C and *Python* programming languages in terms of philosophy, structure, and application. The methodology used was qualitative in nature and employed a bibliographic approach. The C language is compiled, procedural, and statically typed, offering high performance and direct memory control through pointers. *Python*, on the other hand, is interpreted, object-

oriented, and dynamically typed, prioritizing simplicity, readability, and speed in development. While C requires greater technical knowledge and a more detailed syntax, *Python* presents a more concise and easier-to-learn code. C is widely used in low-level systems and applications that require high performance. *Python* excels in areas such as automation, data science, and rapid application development. Another important difference lies in memory management: C requires manual control, while *Python* uses automatic Garbage Collection. Thus, the choice between the two languages depends on the project's needs, balancing performance, simplicity, and productivity.
Keywords: Compiled language, Interpreted language, Performance, Simplicity

RESUMEN
Este artículo describe las principales diferencias entre los lenguajes de programación C y *Python* en cuanto a filosofía, estructura y aplicaciones. La metodología empleada fue de carácter cualitativo y se basó en un enfoque bibliográfico. El lenguaje C es compilado, procedimental y de tipado estático, ofreciendo alto rendimiento y control directo de la memoria mediante punteros. *Python*, por otro lado, es interpretado, orientado a objetos y de tipado dinámico, priorizando la simplicidad, la legibilidad y la velocidad de desarrollo. Si bien C requiere mayores conocimientos técnicos y una sintaxis más detallada, *Python* presenta un código más conciso y fácil de aprender. C se utiliza ampliamente en sistemas y aplicaciones de bajo nivel que requieren alto rendimiento. *Python* destaca en áreas como la automatización, la ciencia de datos y el desarrollo rápido de aplicaciones. Otra diferencia importante radica en la gestión de la memoria: C requiere control manual, mientras que *Python* utiliza la recolección automática de basura. Por lo tanto, la elección entre ambos lenguajes depende de las necesidades del proyecto, buscando un equilibrio entre rendimiento, simplicidad y productividad.
Palabras clave: Lenguaje compilado, Lenguaje interpretado, Rendimiento, Simplicidad

1 INTRODUÇÃO

As Linguagens C e *Python* representam duas das mais influentes ferramentas para o desenvolvimento de programas no mundo da programação (Feltrin, 2019), cada uma com a sua própria filosofia (Schildt, 1996). Uma com o foco maior na manipulação de elementos básicos como bytes e endereços de memória (Backes, 2013) e a outra em contraste, foi projetada com o foco na simplicidade de design, código mais enxuto e menos verboso, sendo frequentemente escolhida por novos programadores (Paiva *et al*, 2019).

Este artigo teve como objetivo detalhar as principais diferenças entre as duas linguagens com abordagem das suas filosofias e modelos de execução, contando com a análise de suas distinções de sintaxe e estrutura onde os diferentes paradigmas e sistemas de tipagem se encontram mais ou menos apropriados para cada situação, temas estes discutidos na disciplina de Paradigmas de linguagem de programação, no curso de Sistemas de Informação, na Unemat, Campus de Sinop - MT.

2 METODOLOGIA

A metodologia utilizada foi de natureza qualitativa e abordagem bibliográfica. A pesquisa qualitativa possibilitou uma análise interpretativa do tema, enquanto o levantamento bibliográfico por sua vez, permitiu reunir e examinar as ideias e informações teóricas e conhecimentos empíricos, visando a compreensão das principais discussões sobre as diferenças cruciais referentes a linguagem C e a linguagem *Python*.

Segundo Gil (2019), a pesquisa qualitativa permite uma investigação mais aprofundada dos fenômenos, considerando seus significados, interpretações e contextos. Dessa forma, o levantamento bibliográfico contribuiu para reunir conceitos teóricos e comparações relevantes acerca dos paradigmas, sintaxe, desempenho e aplicações das duas linguagens de programação.

3 RESULTADOS

A linguagem de programação é de suma importância, pois serve como um universo conceitual na maneira de se pensar em progração. Nada mais é que a forma que permite a expressão e comunicação de ideias entre pessoas e, de forma crucial, facilita a comunição de ideias entre pessoas e computadores. O estudo das linguagens e dos paradigmas (como imperativo, orientado a objetos, funcional e lógico) (Silveira *et al.*, 2021) é essencial para que os estudantes das mais diversas áreas da tecnologia se tornem mais bem informados no futuro (Tucker *et al.*, 2010).

As Linguagens de progração tem sua importância a ser analisada através do impacto intelegcual e dos critérios de projeto que promovem a qualidade do software que foi desenvolvido e dos domínios de aplicação por ela suportados. Dentre eles o poder expressivo e capacidade intelectual e a confiabilidade e qualidade de projeto como dois fatores a serem levados em conta.

No que se refere ao Poder Expressivo e Capacidade Intelectual, as linguagens de programação influenciam diretamente a capacidade intelectual de seu desenvolvedor (Silveira *et al.*, 2021) devido ao poder expressivo que elas carregam, focando na comunicação das ideias computacionais (Tucker *et al.*, 2010). Neste campo se importa a função de Abstração e Redução de Limitações.

Quando se fala em abstração, se trata de uma função crítica das linguagens. O processo de abstração procedural permite aos desenvolvedors a concentração total no que se refere a interface da função e no que se é calculado a partir dela, ignorando outros detalhes de como o cálculo é executado (Tucker *et al.*, 2010). Ela suporta a definição e o uso das estruturas operações complicadas ignorando os detalhes de sua implementação através da abstração dos dados (Silveira *et al.*, 2021).

Por sua vez a Redução de Limitações, refere-se ao conhecimento de uma variadade mais ampla de recursos de linguagens de programação, reduzindo as limitações no desenvolvimento de *apps*. Se uma linguagem não suporta ponteiros, por exemplo, o programdor pode utilizar um vetor para criar uma pilha, contornando a limitação da linguagem (Silveira *et al*., 2021).

No que se refere a Confiabilidade e Qualidade de Projeto, a relevancia se dá de uma linguagem influenciar na confiabilidade do programa (Silveira *et al.*, 2021). Será a medidada em que ele se comporta de abordo com suas especificações e condições (Tucker *et al*., 2010). Destacam-se no referenta a Confiabilidade e Qualidade de

Projeto a verificação de tipos, sintaxe e legiblidade, corretude e ortogonalidade.

A cerca da verificação de Tipos, se trata de um sistema rigoroso que permite a o compilador a execução da verificação antes da execução, reduzindo a chance de erros na execução do código. Essa detecção dos erros acaba por ser crucial, mesmo dita precoce, mas o custo de correção aumenta frasticamente em fases posteriores do desenvolvimento (Tucker *et al.*, 2010).

Já a respeito da Sintaxe e Legibilidade, a sintaxe, ou seja, a forma como o programa é escrito, deve ser simples, afinal as linguagens simples de serem analisadas pelo compilador acabam por serem também simples de serem analisdas pelo programador humano (Silveira *et al.*, 2021). A legibilidade por sua vez, é o que facilita a ler e entneder o programa, facilitando o processo da sua devida manutenção (Tucker *et al*., 2010).

No que tange a Corretude, as linguagens modernas, estão evoluindo para suportar métodos formais, o que permite corretude de programas, provando a mesma matematicamente. A manipulação de exceções também auxilia na confiabilidade, permitindo que o software intercepte erros em tempo de execução, aplicando também as medidadas corretivas. Outro fator que segue essa mesma linha é a Ortogonalidade, que nada mais é onde um pequeno conjunto de construções primitvas podem ser combinadas de muitas maneiras consistentes. É o que dá como resultado menos regras excepcionais (Silveira *et al*., 2021) e leva a programas mais simples e claros (Tucker *et al.*, 2010).

Isso tudo importa, pois, a linguagem de programação se trata do projeto estrutural que define o que pode ser construido e o quão resistente essa construção pode ser (Silveira *et al.*, 2021). Dentre as mais diversas linguagens de programação, hoje analisaremos de forma comparativa duas linguanges diferentes, *Python* e *C*. Linguagens consolidadas no mercado que são extremamente influentes.

Python é uma linguagem de programação e desenvolvimento conhecida pela filosofia de simplicidade no design sendo frequentemente escolhida como uma linguagem de ponto inicial para se introduzir na programação. Sendo criada por Guido van Rossum em 1991 (Feltrin, 2019), tinha o objetivo inicial de permitir um código enxuto, isto é, menos verboso, onde era buscado menos caracteres especiais como em linguagens com sintaxes mais complexas (Paiva *et al.*, 2019).

Por sua vez, a Linguagem C, criada por Dennis Ritchie, no ano de 1972 (Antonello, 2020), é atualmente considerada uma das linguagens de alto nível mais bem-sucedidas e utilizadas de todos os tempos, influenciando diversas linguagens que a antecede. Ao contrário de *Python*, C é classificada como uma linguagem dita processual e muito estruturada, permitindo que problemas complexos sejam decompostos em módulos mais simples, como funções de *if, else, while, switch, for*, entre outras. C também é frequentemente chamada de linguagem de médio nível (Schildt, 1996) por combinar funções como a da linguagem de alto nível com capacidade de realizar coisas como a linguagem *Assembly* faz, manipulando elementos considerados básicos ao computador, como os bytes e endereço de memória, sendo esse o motivo da origem do apelido (Backes, 2013). A linguagem C influenciou o desenvolvimento das mais diversas linguagens, isso inclui a linguagem *Python* (Feltrin, 2019).

No que diz respeito as características da Linguagem *Python* a cerca de modelo de execução e performance é importante destacar que seu modelo de execução se trata de uma linguagem interpretada (Paiva *et al.*, 2019) enquanto a linguagem C tem

seu modelo de execução como Compilada (Schildt, 1996). A cerca da velocidade de desenvolvimento, a linguagem *Python* é rápida para codificar e descomplicada (Feltrin, 2019), enquanto por sua vez a linguagem C é considerada de difícil aprendizado, principalmente pelo conceito de ponteiros (Backes, 2013).

No que cerne ao desempenho, a linguagem *Python* possui diversos recursos para a otimização como as *List Comprehensions*, que podem ser 35% mais rápidas que laços de for (Paiva *et al*., 2019), enquanto por sua vez, a linguagem C, tem o seu compilador que gera códigos mais enxutos e velozes que diversas outras linguagens, sendo extremamente rápida no que tange ao tempo de execução (Antonello, 2020). Por fim, o último ponto que se destaca é a compilação, em que no caso do *Python*, o ambiente de programação pode rodar o código em tempo real de forma nativa (Feltrin, 2019), já, a respeito da linguagem C, a compilação é um processo de múltiplas etapas, incluindo o pré-processamento, a verificação sintática, compilação e *link*- edição (Backes, 2013).

A Delimitação em Blocos na linguagem *Python*, tange a marcação de blocos de código como estruturas condicionais ou de repetição, sendo feitas por meio de identação (Feltrin, 2019). Se observado, o código do *Python* se trata de um bloco de instruções. Em demais linguagens como Java ou C++, um bloco de instruções por sua vez, geralmente é definido por um par de chaves (Paiva *et al.*, 2019).

Python por sua vez, busca ser menos verbosa, o que implica em ser uma linguagem com significativamente uma quantidade menor de caracteres especiais. Uma frase como *Hello World* pode ser escrita de forma simples em *Python* com um simples comando 'print("Hello World")' (Paiva *et al.*, 2019) significativamente mais simples que a linguagem C que precisa de inclusão de bibliotecas (como em #include <stdio.h>), a função 'main()' e o comando 'printf()' (Feltrin, 2019).

Ambas as Linguagens, C e *Python* suportam estruturas condicionais (*if, else*) e estruturas de repetição como (*while, for*) (Feltrin, 2019) (Antonello, 2020). C também inclui outros comandos como o (*do-while*) e o comando de seleção múltipla como o *switch* (Schildt, 1996). *Python* por sua vez, não possui de forma nativa a função *switch*, mas a sua funcionalidade para com o sistema pode ser simulada (Paiva *et al*., 2019).

Tanto a linguagem *Python* quanto a linguagem C são *case sensitive*, ou seja, elas distinguem letras maiúsculas de letras minúsculas. Isso significa que, se utilizar a mesma palavra duas vezes, porém, uma com um caractere maiúsculo e a outra com o mesmo caractere só que em minúsculo, essa palavra é vista como totalmente distinta uma da outra. (Ex.: var1 e Var1 são variáveis distintas) (Feltrin, 2019; Backes, 2013).

Python se destaca como o uma linguagem de tipagem dinâmica, isso significa que o programador não precisa explicitamente declarar o tipo de dado de uma variável, ou seja, uma mesma variável pode armazenar valores de diversos tipos, tipos estes que podem diversificar durante a execução do programa. Além disso, em *Python*, tudo pode ser considerado um objeto (Feltrin, 2019; Paiva *et al*., 2019).

Por sua vez, a linguagem C não é considerada uma linguagem dinâmica, variáveis em C, devem ser declaradas e seus títulos devem ser definidos no momento da declaração, seja o tipo de dados um *char, int, float, double* e *void* (Schildt, 1996).

Python por sua vez não possui uma maneira explicita de declarar uma constante (Paiva *et al*., 2019) se o relacionar a C, que por sua vez, em questão de declaração de constantes pode ser feita utilizando palavras reservadas a mesma, como define ou const (Antonello, 2020). Destaca-se como uma linguagem totalmente Orientada a Objetos (Feltrin, 2019) sendo uma linguagem que suporta naturalmente

os paradigmas de Programação Orientada a Objetos e Programação Funcional. (Paiva *et al*., 2019). A Linguagem C por sua vez é uma linguagem primariamente procedural (Backes, 2013), a funcionalidade de Orientação a Objetos é tipicamente associada a linguagem C++ que é considerada a sua sucessora (Schildt, 1996).

A distinção nos paradigmas de programação entre C e *Python* acaba sendo uma das diferenças mais fundamentais entre ambas linguagens pois refletem filosofias de design e domínio de aplicação. O C se destaca como Procedural e Estruturada e o *Python* pela sua Orientação a Objetos e por ser Multipardigma.

A linguagem C é definida como procedural, significando que ela permite que um problema complexo possa ser decomposto em módulos que neles estão um problema a ser resolvido (Backes, 2013). Essa linguagem também é fortemente associada a Programação Estruturada, pois, esse paradigma é um método disciplinado de escrever programas, fundamentado no uso das estrurutas de controle: sequência (execução de comandos), decisões (também chamado de condições) e iterações (repetições) (Paiva *et al*., 2019). O desenvolvimento de softwares na linguagem C enfatiza o uso dessas técnicas (Deitel; Deitel, 2011), o que colabora para que C seja utilizado no aprendizado para o aprendizado da programação pois a Programação Estruturada acaba sendo, historicamente, muito popular no aprendizado.

Em contraste, a linguagem *Python* foi projetada com a filosofia de haver simplicidade em seu design, visando códigos mais enxutos e com menor verbosidade. Se trata de uma linguagem que suporta uma diversidade de paradigmas, incluindo a Programação Orientada a Objetos (POO), no qual se especializa. Para *Python*, é como se tudo fosse objeto e devido a essa natureza, ela nativamente tem essa orientação, mesmo que o programador deseje trabalhar com o paradigma da programação estruturada. As próprias variáveis são objetos (Feltrin, 2019). Destaca-se a abstração e reuso como característica dessa linguagem, afinal, a Orientação a Objetos acaba visando uma programação que por essencia, é mais voltada ao mundo real por meio da abstração. As classes, o conceito central da POO, tem a atuação como um molde para a criação de objetos, permitindo de forma eficiênte a reutilização do código (Paiva *et al*., 2019).

Em suma, enquanto a linguagem C foca na estrutura procedural para controle de baixo nível, a linguagem *Python* permite uma abstração total, onde todos os elementos se tornam objetos, o que a torna intrisecamente orientada a objetos, mesmo que ela permita suporte a outros paradigmas, fora a POO.

A Linguagem C é valorizada pela sua capacidade de oferecer um acesso de baixo nível a memória, o que rende a ela o apelido de linguagem de médio nível (Antonello, 2020). Isto permite a implementação de programas com instruções em Assembly, uma linguagem propriamente de baixo nível, sendo muito útil quando a dependência do tempo de processamento é crucial ao funcionamento do programa (Backes, 2013). C utiliza de forma intensiva o conceito de ponteiros, que apesar de ser considerado por muitos de difícil aprendizado, são variáveis que armazenam endereços de memória (Schildt, 1996). *Python*, por sua vez, conta com um gerenciador de memória automático chamado *Garbage Collector* (Coletor de Lixo) o que contribui para a facilidade do uso, pois o programador não precisa se preocupar em gerenciar o uso da memória manualmente (Paiva *et al*., 2019).

Na linguagem C, trabalha com tipos básicos como *chat, int, float, double e void* (Schildt, 1996). Para formular estruturas de dados mais complexas, o programador acaba por utilizar comandos para definir novos tipos, um exemplo

prático se dá na definição de Strings, que são implementadas como uma espécie de vetor de caracteres. A modularidade se dá pela inclusão de bibliotecas estáticas, pela diretiva #include (Antonello, 2020), e criação de funções (sub-rotinas). É possível ao programador desenvolver suas próprias bibliotecas personalizadas (Backes, 2013), estas que são incorporadas ao programa usando o link-edição (Schildt, 1996).

Por sua vez, a linguagem *Python* em contraste, vem com o necessário no funcionamento (*batteries included*), se destacando por ser chamada de pronta para o uso (Feltrin, 2019). Oferece uma grande variedade de funcionalidades pré-carregadas. Os dados complexos não são manipulados como *arrays* básicos, mas sim como *built-ins* e objetos, sendo *list* (listas) e *dict* (dicionários) para armazenar conjuntos de elementos (Paiva *et al*., 2019). Beneficia de uma imensidão de módulos de extensão e bibliotecas inteiras prontas para o uso, especialmente para áreas como *data science* e *machine learning*. Isso permite um desenvolvimento mais rápido e com melhor performance no que tange o uso da memória. Isso otimiza o código (Feltrin, 2019).

Na linguagem C, uma *string* acaba por ser definida como um *array* de caracteres (*char*) (Backes, 2013), que resulta no fato de que, se quiser usar funções de manipulação de *strings* (tamanho, cópia e comparação), são implementadas através das bibliotecas, como o *string.h* (*strlen(), strcpy(), strcmp()*, sendo esse último, em C, um *case-sensitive*) (Antonello, 2020) (Schildt, 1996), enquanto na linguagem *Python*, *str* é um tipo de dado *built-in* e a concatenação de *strings* pode ser feita utilizando operadores de adição + (Feltrin, 2019).

Em relação aos valores lógicos para *Python* podem ser *True* ou *False*. True representado por 1. False representado por 0 (Paiva *et al*., 2019), já em C, tudo aquilo que é um valor diferente de zero é considerado verdadeiro, o zero é considerado falso. O padrão C99 adicionou o tipo *_Bool* para compatibilidade (Backes, 2013).

4 CONCLUSÃO

Em síntese, existem diferenças fundamentais entre as linguagens C e *Python*, que embora exista relação entre elas, afinal como antes dito, C influenciou grandemente a criação do *Python*, ambas as linguagens atendem filosofias e necessidades distintas (Schildt, 1996) (Feltrin, 2019).

A Linguagem C se firma como uma linguagem que é conceito como linguagem procedural, compilada, dita como tipagem estática (Schildt, 1996), com o seu diferencial maior a performance de execução e acesso ao baixo nível da memória por utilizar ponteiros (Backes, 2013), o que rende a ela o apelido de linguagem de médio nível (Antonello, 2020).

Por outro lado, a linguagem *Python* se posiciona como uma dita linguagem interpretada, orientada a objetos de linguagem dinâmica. Ela atende a filosofia de um design simples e legível, resultando em menor verbosidade com um desenvolvimento mais rápido e fluído (Feltrin, 2019). Utiliza do *Garbage Collector* para abstrair a complexidade de lidar com a utilização da memória (Paiva *et al*., 2019).

A escolha entre as duas linguagens é totalmente dependente do objetivo do projeto, sendo C ideal para cenários que exijam, máximo desempenho de execução e controle de hardware (Antonello, 2020), enquanto *Python* tem o seu destaque atrelado a sua velocidade no desenvolvimento e facilidade de seu uso (Paiva *et al*.,

2019).

REFERÊNCIAS

ANTONELLO, Ricardo. **Linguagem C** E-book. 1ª Ed. 2020. Disponível em: https://antonello.com.br/livro-linguagem-c/. Acesso em: 20 out. 2025.

BACKES, André. **Linguagem C:** Completa e Descomplicada. 1ª Ed. São Paulo: Elsevier Editora Ltda, 2013.

DEITEL, Paul J.; DEITEL, Harvey M. **Como Programar em C.** 6ª edição. São Paulo: Pearson Universidades, 2011. 846 p.

FELTRIN, Fernando. ***Python* do ZERO à programação orientada a objetos**. Curitiba: Uniorg, 2019.

GIL, Antonio Carlos. **Métodos e técnicas de pesquisa social.** 7. ed. São Paulo: Atlas, 2019.

SILVEIRA, Sidnei Renato; DELEPIANE DE VIT, Antônio Rodrigo; BERTOLINI, Cristiano; PARREIRA, Fábio José; CUNHA, Guilherme Bernardino da; BIGOLIN, Nara Martini. **Paradigmas de programação:** Uma introdução. Belo Horizonte: Synapse Editora, 2021. 95 p. Disponível em: https://doi.org/10.36599/editpa-2021_ppui. ISBN 978-65-88890-08-0.

PAIVA, Fábio Augusto Procópio de; NASCIMENTO, João Maria Araújo do; MARTINS, Rodrigo Siqueira; SOUZA, Givanaldo Rocha de. **Introdução a *Python*:** com aplicações de sistemas operacionais. Natal: IFRN, 2019.

SCHILDT, Herbert. **C: completo e total**. 3ª Ed. São Paulo: Markrom Books Ltda, 1996.

TUCKER, Allen B.; NOONAN, Robert E. **Linguagens de programação:** princípios e paradigmas. Porto Alegre: AMGH, 2010.