

# Comparing and Contrasting of C and Python Programming Languages

October 2023, Tugay Talha İçen

## 1. Introduction

### 1.1. Overview of C and Python

C and Python are the pretty popular and widely used programming languages in the programming world. C is a procedural language and it is created in the early 1970s by Dennis Ritchie at Bell Labs. Also it is a low-level language, people prefer for its efficiency and performance. On the other hand Python is one of the scripting languages which released in the 1990s (public release 1994) from the hand of Guido van Rossum. On the contrary C, Python is a high-level language, people uses for its simplicity and readability.

Due to its low memory and processing power usage, C is still used in embedded systems and operating systems nowadays. On the other side Python has very largely uses even in embedded systems but mainly used in developing websites and software, task automation, data analysis, data visualization, machine learning, and Blockchain.

### 1.2. Historical Context and Emergence

Before the invention of C, there were a few languages that laid the groundwork. BCPL (Basic Combined Programming Language) is one of the earliest. Developed by Martin Richards in the mid-1960s, BCPL was typeless and was designed for writing compilers and operating systems. We can see the BCPL's influence in the syntax and semantics of C.

Developers desired portability and to enhance the programming environment, then higher-level language provided these. The objective was have a language close enough to machine code (low-level)

but with features allowed for easier to code (Of course C for those days...). Dennis Ritchie took the principles from BCPL and added type checking, which became the cornerstone of the C language. Introduced in the early 1970s, C provided a incredible unique coalescence of low-level access to memory and the a high-level language features. This paved the way for operating systems to be carried more easily on different hardware and written in C.

If we look at the contrast, Python confronts to us as both versatile and user-friendly programming language. Its development can be traced back to the late 1980s and early 1990s under the aegis of Guido van Rossum, a Dutch computer scientist. Van Rossum, dissatisfied with the constraints of the ABC language, embarked on creating Python. The ABC language, although pioneering in its simplicity and educational focus, was somewhat hamstrung in terms of its flexibility and power.

Python takes it's name from British comedy series "Monty Python's Flying Circus," Python designed with two primary tenets in mind: the simplicity and clarity reminiscent of ABC, and an augmented power and adaptability that its predecessor lacked.

In 1994, when Python was first released to the public, it was met with burgeoning interest. Its concentration of simplicity and power rendered it an instant favorite among programmers. The code's readability, facilitated by its reliance on indentation and clear syntax, made it especially captivating.

One of Python's most salient features is its interpreted nature, eschewing the need for cumbersome compilation processes and thereby streamlining code execution. This interpretive quality, and high-level design, positions Python as an optimal choice for rapid development and prototyping. Another quintessential trait of Python is its dynamic typing system. By absolving programmers from the need to declare variable types explicitly, it offers a more fluid and organic coding experience.

But of course, this two geriatric programming languages(C and Python) evoluated a lot since their release. C has updated to support new features such as object-oriented programming and generic programming. Python also updated to support asynchronous programming and type hints. And we will discuss these issues later as well.

Then, it is time to compare two languages in more detail and with the programming languages paradigms.

## 2. Paradigmatic Features

### 2.1. C: The Pantheon and The Epitome of Procedural & Imperative Programming

C is a procedural programming language. Procedural programming languages are designed to solve problems by breaking them down into smaller, step-by-step procedures. Procedural programming is a traditional approach to programming, and it is still widely used today for a variety of tasks, such as system programming, embedded systems programming, and performance-critical applications.

Because each step allows people to define their own desires well and this makes it efficient for low-limited systems. On the other part declarative systems accomplish your task with less instructions but follow through more unnecessary service for your case to be done more things with same command.

Here it is an example procedural (imperative type& written in C) and an example for declarative code(written in SQL), Procedural one:

```

5 Write your code in this editor and press "Run" button
6
7 *****
8
9 #include <stdio.h>
10
11 int main() {
12     int numbers[] = {1, 2, 3, 4, 5};
13     int sum_of_squares = 0;
14     int size = sizeof(numbers) / sizeof(numbers[0]);
15
16     for(int i = 0; i < size; i++) {
17         sum_of_squares += numbers[i] * numbers[i];
18     }
19
20     printf("%d\n", sum_of_squares);
21
22     return 0;
23 }
24

```

And the declarative one:

```

Write your code in this editor and press

*****
SELECT SUM(value * value) FROM numbers;

```

As we can see the declarative one is much shorter and black box and relatively easier to develop than the procedural program, but we don't know the process in behind. In the our other option, procedural, flow is in developers hand and more optimizable depend on situation.

In C, programs can be written as a series of functions. Each function does a specific task, and functions can be called from other functions. This allows programmers to modularize their code, making it more reusable, maintainable, and much debugable.

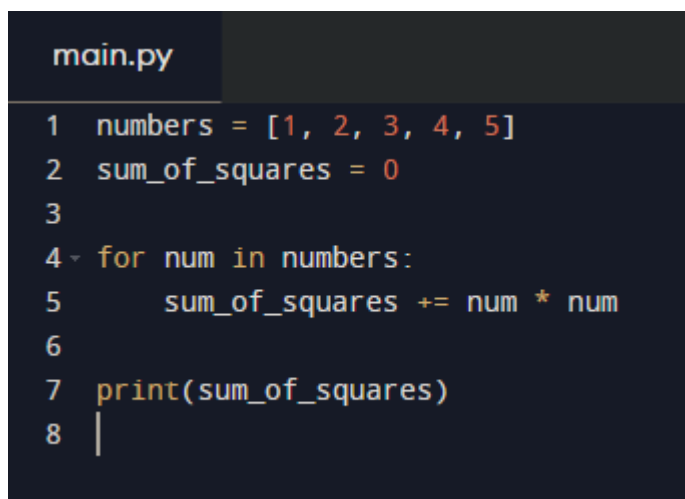
Another key feature of C is its direct access to hardware and memory. This gives C programmers a high degree of control over their programs, but it also makes C programming more complex (therefore hard to learn & use) and error-prone.

## 2.2. Python: A Multi-Paradigm Voltron

Python is a very high-level language, which bounteously combines great numbers of programming paradigms. In our story it will form Voltron with 4 paradigm (legs and arms) and 1 devoloper (torso-head)

### 2.2.1 Python's. Imperative & Procedural Flame – Red Lion

Imperative programming is one fo the software paradigms. It employs statements to modify the state of a program as we talked about in C part too. Functions are implicitly coded at every step to solve a problem in imperative programming, if you don't use pre-coded models of corz. Samet ask as above as imperative-procedural in Python:



```

main.py
1  numbers = [1, 2, 3, 4, 5]
2  sum_of_squares = 0
3
4  for num in numbers:
5      sum_of_squares += num * num
6
7  print(sum_of_squares)
8  |

```

As we can see it is pretty much similar with C as imperative.

### 2.2.2. Python's Object Oriented Essence – Green Lion

In Object Oriented Programming, everything consists of objects. So what is this object? We can say that objects are representatives or examples of existing or constructed structures they (mostly called as object, types or classes). Undoubtedly, Python is one of the languages that have taken its essence from this green lion. These features allows to build reusable, sharable, flexible, secure codes. Example code for object oriented programming in Python:

```
main.py
1 class NumberList:
2     def __init__(self, numbers):
3         self.numbers = numbers
4
5     def sum_of_squares(self):
6         return sum(num * num for num in self.numbers)
7
8 numbers_list = NumberList([1, 2, 3, 4, 5])
9 print(numbers_list.sum_of_squares())
10 |
```

In the first aspect it doesn't seem like worthwhile, but when we think about it more broadly, as we mentioned above, it provides numerous advantages. If we need to list them, we can implement many new methods to this class very easily. We can save it and import it into other codes to use it repeatedly. We can even develop a library like NumPy by expanding it with new methods, and make it available to people. We can also improve our code with their support and let them benefit from it.

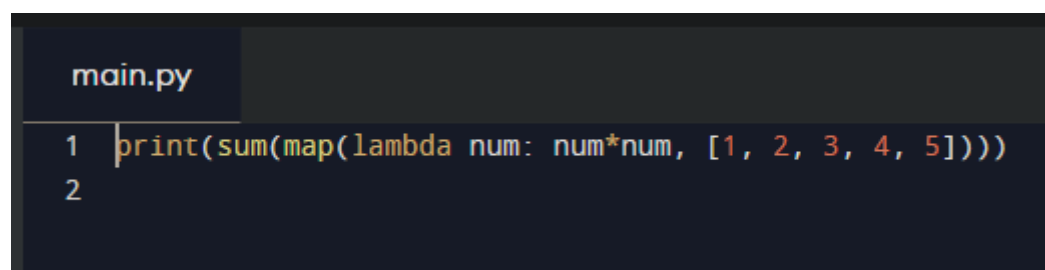
### 2.2.3 Functional Power – Yellow Lion

Functional programming is a programming paradigm based on the programs should be written as pure functions that takes inputs and return outputs idea, without modifying their state or the state of the world around them (datas). If a function change a variable from outside it makes the function unpure function. This paradigm makes the code more readable, more robust to errors, and less

challenging to debug. Furthermore, functional programming furnishes code with more benefits like parallelism, reusability and composability, but we will not go into the details of these in this article.

#### 2.2.4 Declarative Side of The Ocean – Blue Lion

Declarative programming is one of the other paradigms. On the contrary to imperative programming in this paradigm, instead of telling the program how to do something step-by-step, we only tell it what to do, and the programming language does it for us. This makes coding easier, but on top of that it causes a loss of control over the code. Let's give an example for functional & declarative as well to make it more explanatory, since functional declarative programming is a subcategory of it, we are giving an example of both of them here:



```
main.py
1 | print(sum(map(lambda num: num*num, [1, 2, 3, 4, 5])))
2 |
```

As we can see in the photo and as we have mentioned before, our code is much shorter in this type of programming. We do not use any external variables and we leave everything to pure functions. This is very helpful for us to be able to make cumulative progress while writing higher-level programs.

#### 2.2.5 Developer, The Last and Most Important Part – Black Lion

As expected, this is not part of Python, but just as Voltron cannot be Voltron without a head, a developer can create a program without a programming language without a programming language. Just as Python is more user-friendly than C, which replaced BCPL, Python also shows itself as a more user-friendly language than C, even though it gives the developer less control over memory. With its wide range of libraries, it allows developers to do what they want in a way that is easy or in a way that they want, such as imperative or declarative, OOP or non-OOP. And if you want to check out our Developer example, you can also check out the following links :)) :

<https://www.linkedin.com/in/tugay-talha-i%C3%A7en-4793461a2/>

<https://www.hackerrank.com/profile/tugaytalha>

<https://github.com/Tugaytalha>

Looking at those examples, it's like the developer's magic trick: turning code into an actual program.

Ta-da! 🎩 ✨.

### 3. Syntactical - Lexical Analysis

#### 3.1. Tokens and Keywords: A Closer Look

I think it would be a good idea to start by defining what tokens and keywords are. Token in programming languages is the basic component of the source code which makes it easier to understand for the computer. There are five types of tokens, these are constants (literals), identifiers, operators, separators (punctuation), and reversed words (~keywords, all keywords are reversed words but not the opposite).<sup>1</sup> Keywords are pre-defined expressions for special use in a programming language.

##### 3.1.1. Tokens and Keywords In C

C is one of the languages that run under compiler, it takes the codes as a text then turn them into tokens. After it turns the code into tokens it prepares the machine code according to these tokens. Let's talk about token types in short:

---

<sup>1</sup> <https://www.capterra.com/glossary/token/>

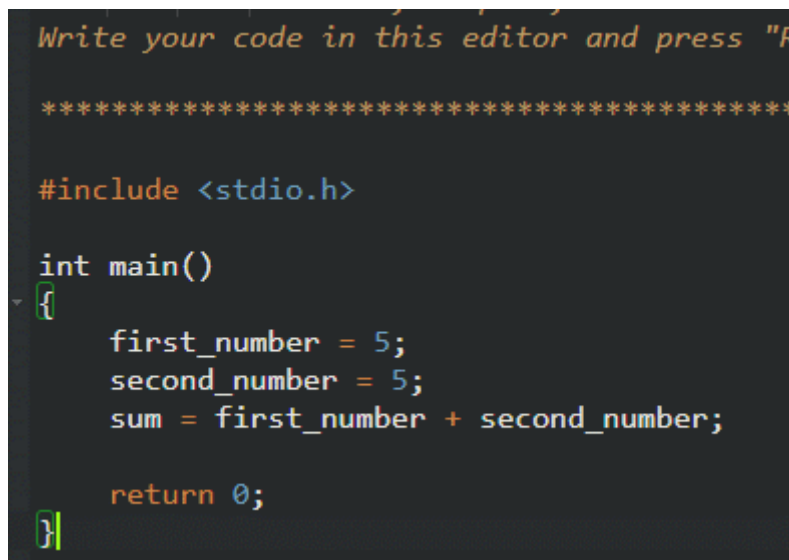


### 3.1.1.1 Identifiers In C

Identifiers is short, informative and unique quilt for variables and functions. These names are defined by users. Rules for identifiers in C:

- First character of identifier must be a letter or underscore
- Identifiers are case sensitive, e.g “names” and “Names” are completely different identifier
- Identifiers can only consist of letters, underscores, and digits.
- Identifiers cannot have same name with keywords (we will talk about keywords in text titles)

In this code first\_number, second\_number and sum are examples for keywords:



```

Write your code in this editor and press "R"

*****

#include <stdio.h>

int main()
{
    first_number = 5;
    second_number = 5;
    sum = first_number + second_number;

    return 0;
}

```

### 3.1.1.2 Constants In C

Constants are the variables that have unchangeable values. We can say there are 7 types of constants in C, 4 for numerical (Integer, floating point, octal, hexadecimal) values, 2 for alphabetical (string, character), and 1 for logical (true or false). Constant can be defined by the user with using “const” keyword or “#define”.

### 3.1.1.3 Separators (Punctuation) In C

Separators are special characters that have separate missions like reaching an index an array (square brackets) in C. These separators are square brackets ([]), simple brackets (()), curly braces ({}), comma (,), hash (#), asterisk (\*), tilde (~), colon (:), and semi colon (;). Click [here](#) to see its functions.

### 3.1.1.4 Keywords In C

As we have already discussed, keywords are predefined words with specific functions. These are cannot usable as variable name and contains only lower-case letters in C. There are [32 keywords](#) in C. Most important ones are type keywords which is using to define a type, break, case, continue, else, enum, if, while, struct, and switch. You can also use typedef for define your type like int.

### 3.1.1.5 Operators In C

An operator is a specific character to represent a mathematical or logical process between constants or identifiers. There are 7 types and 38 different operators (could change depend on source) [operators in C](#). I am soo tired. If you do it enviably, this homework is really exhausting.

## 3.1.2 Tokens and Keywords In Python

Naturally, Python also has the same 5 tokens as C (constants (literals), identifiers, operators, separators (punctuation), and keywords). That's why we'll focus on the differences rather than explaining it in depth again.

### 3.1.2.1 Keywords In Python

Python and C keywords function similarly, and [Python has 35 different keywords](#), many of which are similar. The key difference here is that C works with a compiler, while Python works with an interpreter. C compiles the code as a whole before running it, while Python compiles it piece by piece during runtime.

### 3.1.2.2 Constants In Python

Different between Python and C constant is Python has no separate char type constant, and has a None constant which is using for nothings. And the other difference in depth is C perceive all constant as integer and allows to use interchangeably but the python do not allow directly to do this. Additionally, there is no “const” keyword and “#define” in Python.

### 3.1.2.3 Operators In Python

Python has logical operators as a string and also has 2 more string operator types which is membership (in, not in) and the identity (is, is not). I personally believe that these extra operators are functional operators that add quality of life, even if they are not efficient.

## 3.2. Readability and Concision: C vs Python

This is a short point with not much to talk about. Although C has made some efforts to be more user-friendly, its focus on efficiency and control on the other hand Python has a syntax that is more similar to natural language and requires more structured code as an interpreted language, this situation makes Python more readable than the C.

In the lexical part, two languages have similar structures but the Python doesn't uses type names in general and has a little more lexical parts than the C to provide more functionality.

## 4. Some Semantical Comparisons

Semantics defines what a computer program does when it is executed. About difference between syntax, syntax is about the structure and format of an expression, while semantics is about the meaning of the expression. Syntax errors are found during compilation, while semantic errors are found during execution.

### 4.1. Type Systems: Static vs. Dynamic

Type systems define the types of data in a programming language and the operations that can be performed on them. It can be static as C or dynamic as Python. In Static type systems the types of variables must be declared at compile time and cannot change easily. On the flip side Dynamic typing system determines type of the variable at the runtime. This allows for shorter, more readable code and flexibility in variable usage, but it can also lead to various errors due to mismatches at runtime.

### 4.2. Control Structures

Python and C has pretty much similar control structures except C's separate "do-while" loop and Python's "pass" and separate "elif". These common control structures are if, else, for, while, break, continue. Python's control statements works with indentation and C's are working with curly braces. The other important difference is Python's for loop working with any iterable easily due to "in" keyword.

### 4.3. Exception Handling

Python has internal exception error handling system with its try-except structure. In C there is no internal system for exception handling. It can be done manually, or errors can be handled using an external system called gdb on the terminal.

## 5. Additional Considerations

### 5.1. Availability and Ecosystem

Python and C are 2 programming languages that are widely available in most of the operating systems. Both languages are pre-installed on most macOS and Linux distributions, and they are available as a free download for Windows and other operating systems. Both languages have large ecosystems with lots of libraries and frameworks. We can say Python's community is more active and its ecosystem is more diverse than C's nowadays. The reason for this diversity is that Python's focus is extremely broad, while the focus of the C ecosystem is more on operating systems and embedded systems.

### 5.2. Efficiency and Performance

Just saying "C" would be enough.

### 5.3. Learning Curve: Steppingstones for Beginners

Python is generally considered to be a more beginner-friendly language than C. It has a simpler syntax, and it is more forgiving of errors. On the other side C has a lower-level syntax, and it is more unforgiving of errors. As a learner of both of those, the early stages of learning C was quite painful against Python. As we mentioned when discussing syntax, the fact that Python's syntax is closer to natural language is also one of the factors that facilitate learning for beginners.

## 6. Conclusion

Python is a flexible structured, relatively easy-to-learn programming language. Python is both declarative and imperative language that gives users more control over their coding style and it is. C is a more difficult language to learn with an imperative structure that totally gives ropes to users. When you wanna choose between the C and Python, You should consider what they will be used for. C is a good choice for learning the principles of computer science, For applications where efficiency and performance are paramount like operating systems or embedded systems, or applications requires additional memory management. Python is a good choice for hobbies and general-purpose use, for data-related work, or for web development.