

Python Syntax Checker tool for Programmers

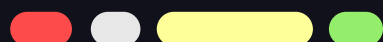
< Project 2 HCL GUVI Internship >
< Mohit Sharma (mohit_2312res895@iitp.ac.in) >





Table of contents

01 Introduction



02 Need for a Syntax Checker

03 Code and its working

04 Conclusion



Introduction

A **Python syntax checker** is a tool that analyzes your Python code to identify and report syntax errors and other issues. These tools, often called **linters**, help you catch mistakes early in the development process without having to run the code. They check for things like missing parentheses, incorrect indentation, and misspelled keywords.



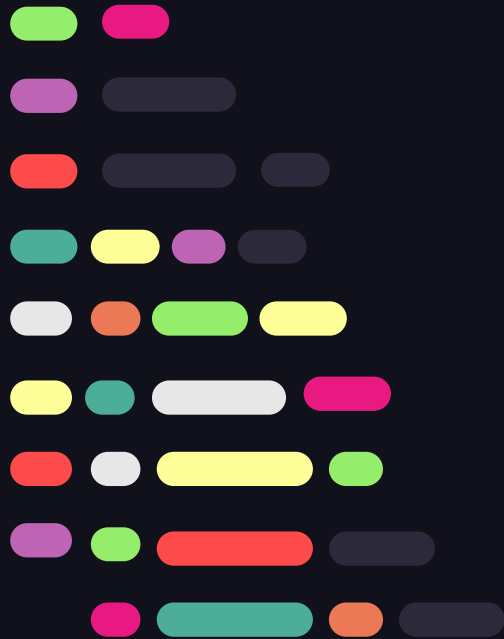


Need for a Syntax checker



- We need a Python syntax checker tool to **improve code quality, catch errors early, and enforce coding standards.**
- It works as an automated proofreader for your code, preventing simple mistakes that can cause a program to crash. Instead of finding out your code doesn't work after trying to run it, a syntax checker highlights errors as you type. This saves time and frustration, especially for beginners.





- Beyond just finding errors, many of these tools (called linters) also help developers write clean, readable code that is easy for others to understand and maintain. This is crucial in team environments where multiple people work on the same project. In short, a syntax checker is a key tool for writing more reliable and professional code.



Code

```
import sys

def check_python_syntax(file_path):
    """
    This Checks for syntax errors in a Python file without executing it.

    Args:
        file_path (str): The path to the Python script.

    Returns:
        None
    """
    try:
        with open(file_path, 'r', encoding='utf-8') as f:
            source_code = f.read()

        # Using the built-in compile() function to validate syntax.
        # The 'exec' mode is used for compiling module-level code.
        compile(source_code, file_path, 'exec')

        print(f"✅ The Python script '{file_path}' is valid and has no syntax errors.")
```

```

except FileNotFoundError:
    print(f"✖ Error: The file '{file_path}' was not found. Please check the file path.")
    sys.exit(1)
except SyntaxError as e:
    print(f"✖ Syntax Error found in '{file_path}' at line {e.lineno}:")
    print(f"    {e.msg}")

# The following lines of code will help the user in identifying the line with the error.
lines = source_code.splitlines()
if e.lineno <= len(lines):
    print(f"    --> {lines[e.lineno - 1].strip()}")
except Exception as e:
    print(f"✖ An unexpected error occurred: {e}")

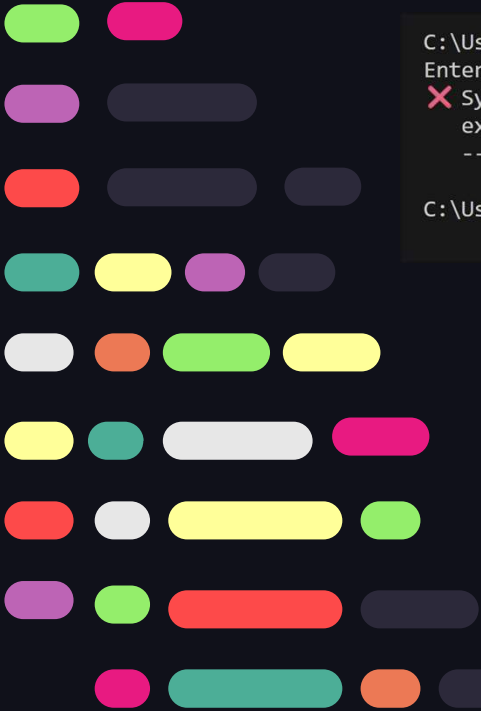
if __name__ == "__main__":

    # Asking the user to enter the file path.
    file_to_check = input("Enter the path to the Python script you want to check: ")
    check_python_syntax(file_to_check)

```



Output



```
C:\Users\mohit>python -u "c:\Users\mohit\Desktop\Python Syntax Checker\syntax_checker.py"
Enter the path to the Python script you want to check: C:\Users\mohit\Desktop\Student_grades_project\student_grades.py
✗ Syntax Error found in 'C:\Users\mohit\Desktop\Student_grades_project\student_grades.py' at line 12:
  expected ':'
    --> for i in range(3)

C:\Users\mohit>
```

When a code path with error is given to the syntax checker tool it instantly showed the error and also pinpointed the line in which the error was present.





Working of the code



The provided Python script fulfills all the functional requirements outlined in the project description. Here's a breakdown of its key components:

- 1. File Input:** The script prompts the user to enter the file path via `input()` . This makes it interactive and easy to use.
- 2. File Handling:** The `with open(...)` statement safely reads the entire content of the specified file into the `source_code` variable. This approach ensures that the file is automatically closed even if an error occurs.



3. Syntax Validation: The core of the tool is the `compile()` built-in function. This function takes the source code, the file path, and a "mode" as arguments. By setting the mode to 'exec', Python attempts to compile the code as a complete program, effectively checking for syntax errors without running any of the code.

Error Handling: A try-except block is used to catch potential errors.

- `FileNotFoundError`: This exception is caught if the user enters a path to a file that doesn't exist.
- `SyntaxError`: This is the most crucial part. If the `compile()` function finds a syntax error it raises a `SyntaxError` exception. The except block catches this and allows us to access useful information like the error message (`e.msg`) and the line number (`e.lineno`) where the error occurred.

4. Displaying Results: The script provides clear, user-friendly feedback. If no `SyntaxError` is raised, it prints a success message. If a `SyntaxError` is caught, it prints the specific error message along with the line number helping the user pin point the problem.





Conclusion

In conclusion, the **Python Syntax Checker Tool** successfully addresses the challenge of identifying syntax errors for new programmers. By leveraging Python's built-in `compile()` function within a robust ***try-except*** block, this project provides a simple yet effective way to validate Python scripts without the need for execution. The tool's ability to catch a ***SyntaxError*** and display the exact line number and error message offers invaluable feedback, allowing learners to quickly pinpoint and correct their mistakes. Ultimately, this project serves as a practical, lightweight utility that streamlines the debugging process, reduces frustration, and empowers beginners to write syntactically correct code more efficiently.

