Pegar en Main.py

```
# Editor Code Component
class DartEditorCode(QPlainTextEdit):

    class NumberBar(QWidget):
        def __init__(self, editor):
            QWidget.__init__(self, editor)

            self.editor = editor

self.editor.blockCountChanged.connect(self.updateWidth)

self.editor.updateRequest.connect(self.updateContents)
            self.font = QFont()
            self.numberBarColor = QColor("#2F2F36")

        def paintEvent(self, event):

            painter = QPainter(self)
            painter.fillRect(event.rect(),
self.numberBarColor)

            block = self.editor.firstVisibleBlock()

            while block.isValid():
                blockNumber = block.blockNumber()
                block_top = self.editor.blockBoundingGeometry(

block).translated(self.editor.contentOffset()).top()

                if not block.isVisible() or block_top >=
event.rect().bottom():
                    break

                if blockNumber ==
self.editor.textCursor().blockNumber():
                    self.font.setBold(True)
                    painter.setPen(QColor("#2F2F36"))
                else:
                    self.font.setBold(False)
                    painter.setPen(QColor("#2F2F36"))
                painter.setFont(self.font)

                paint_rect = QRect(0, block_top, self.width(),

self.editor.fontMetrics().height())
                painter.drawText(paint_rect, Qt.AlignRight,
str(blockNumber+1))

                block = block.next()

            painter.end()

            QWidget.paintEvent(self, event)

        def getWidth(self):
            count = self.editor.blockCount()
            width = self.fontMetrics().width(str(count)) + 10
            return width

        def updateWidth(self):
            width = self.getWidth()
            if self.width() != width:
                self.setFixedWidth(width)
                self.editor.setViewportMargins(width, 0, 0, 0)

        def updateContents(self, rect, scroll):
            if scroll:
                self.scroll(0, scroll)
            else:
                self.update(0, rect.y(), self.width(),
rect.height())

            if rect.contains(self.editor.viewport().rect()):
```

```python
                fontSize =
self.editor.currentCharFormat().font().pointSize()
                self.font.setPointSize(fontSize)
                self.font.setStyle(QFont.StyleNormal)
                self.updateWidth()

    def __init__(self, DISPLAY_LINE_NUMBERS=True,
HIGHLIGHT_CURRENT_LINE=True):
        super(DartEditorCode, self).__init__()

        self.setFont(QFont("Ubuntu Mono", 11))
        self.setLineWrapMode(QPlainTextEdit.NoWrap)

        self.DISPLAY_LINE_NUMBERS = DISPLAY_LINE_NUMBERS

        if DISPLAY_LINE_NUMBERS:
            self.number_bar = self.NumberBar(self)

        if HIGHLIGHT_CURRENT_LINE:
            self.currentLineNumber = None
            self.currentLineColor = QColor("#FDFDFD")

self.cursorPositionChanged.connect(self.highligtCurrentLine)

    def resizeEvent(self, *e):

        if self.DISPLAY_LINE_NUMBERS:
            cr = self.contentsRect()
            rec = QRect(cr.left(), cr.top(),
                        self.number_bar.getWidth(),
cr.height())
            self.number_bar.setGeometry(rec)

        QPlainTextEdit.resizeEvent(self, *e)

    def highligtCurrentLine(self):
        newCurrentLineNumber = self.textCursor().blockNumber()
        if newCurrentLineNumber != self.currentLineNumber:
            self.currentLineNumber = newCurrentLineNumber
            hi_selection = QTextEdit.ExtraSelection()

hi_selection.format.setBackground(self.currentLineColor)
            hi_selection.format.setProperty(
                QTextFormat.FullWidthSelection, True)
            hi_selection.cursor = self.textCursor()
            hi_selection.cursor.clearSelection()
            self.setExtraSelections([hi_selection])

# Label Code Component
class CodeLabel(QWidget):
    def __init__(self):
        super(CodeLabel, self).__init__()
        layout = QHBoxLayout()
        label1_txt = QLabel()
        label1_txt.setText("<h4>Write or Paste your copy here:
</h4>")
        layout.addWidget(label1_txt)
        self.setLayout(layout)

# Print Label Component
class PrintLabel(QWidget):

    def __init__(self):
        super(PrintLabel, self).__init__()
        vb = QVBoxLayout()
        hb_layout = QHBoxLayout()
        label_text = QLabel()
        plain_text = QPlainTextEdit()

        label_text.setText("Execution <strong> Result Analysis
</strong>")
        label_text.setStyleSheet("color: #2D2D2D;")

        plain_text.setReadOnly(True)
        plain_text.setStyleSheet("background-color: #E5E8ED;")

        hb_layout.addWidget(label_text)
        hb_layout.addStretch(1)

        vb.addLayout(hb_layout)
        vb.addWidget(plain_text)
        self.setLayout(vb)
```

CODE

```
from PyQt5.QtWidgets import *
from PyQt5.QtCore import *
from PyQt5.QtGui import *
from lexico import *
from sintactico import *
from errorHandle import *

# Editor Code Component
class DartEditorCode(QPlainTextEdit):

    class NumberBar(QWidget):
        def __init__(self, editor):
            QWidget.__init__(self, editor)

            self.editor = editor

self.editor.blockCountChanged.connect(self.updateWidth)

self.editor.updateRequest.connect(self.updateContents)
            self.font = QFont()
            self.numberBarColor = QColor("#2F2F36")

        def paintEvent(self, event):

            painter = QPainter(self)
            painter.fillRect(event.rect(),
self.numberBarColor)

            block = self.editor.firstVisibleBlock()

            while block.isValid():
                blockNumber = block.blockNumber()
                block_top = self.editor.blockBoundingGeometry(

block).translated(self.editor.contentOffset()).top()
```

```
                if not block.isVisible() or block_top >=
event.rect().bottom():
                    break

                if blockNumber ==
self.editor.textCursor().blockNumber():
                    self.font.setBold(True)
                    painter.setPen(QColor("#2F2F36"))
                else:
                    self.font.setBold(False)
                    painter.setPen(QColor("#2F2F36"))
                painter.setFont(self.font)

                paint_rect = QRect(0, block_top, self.width(),

self.editor.fontMetrics().height())
                painter.drawText(paint_rect, Qt.AlignRight,
str(blockNumber+1))

                block = block.next()

            painter.end()

            QWidget.paintEvent(self, event)

        def getWidth(self):
            count = self.editor.blockCount()
            width = self.fontMetrics().width(str(count)) + 10
            return width

        def updateWidth(self):
            width = self.getWidth()
            if self.width() != width:
                self.setFixedWidth(width)
                self.editor.setViewportMargins(width, 0, 0, 0)
        def updateContents(self, rect, scroll):
            if scroll:
                self.scroll(0, scroll)
            else:
                self.update(0, rect.y(), self.width(),
rect.height())

            if rect.contains(self.editor.viewport().rect()):
                fontSize =
self.editor.currentCharFormat().font().pointSize()
                self.font.setPointSize(fontSize)
                self.font.setStyle(QFont.StyleNormal)
                self.updateWidth()

    def __init__(self, DISPLAY_LINE_NUMBERS=True,
HIGHLIGHT_CURRENT_LINE=True):
        super(DartEditorCode, self).__init__()

        self.setFont(QFont("Ubuntu Mono", 11))
        self.setLineWrapMode(QPlainTextEdit.NoWrap)

        self.DISPLAY_LINE_NUMBERS = DISPLAY_LINE_NUMBERS

        if DISPLAY_LINE_NUMBERS:
            self.number_bar = self.NumberBar(self)

        if HIGHLIGHT_CURRENT_LINE:
            self.currentLineNumber = None
            self.currentLineColor = QColor("#FDFDFD")

self.cursorPositionChanged.connect(self.highligtCurrentLine)

    def resizeEvent(self, *e):

        if self.DISPLAY_LINE_NUMBERS:
            cr = self.contentsRect()
            rec = QRect(cr.left(), cr.top(),
                        self.number_bar.getWidth(),
cr.height())
            self.number_bar.setGeometry(rec)

        QPlainTextEdit.resizeEvent(self, *e)

    def highligtCurrentLine(self):
        newCurrentLineNumber = self.textCursor().blockNumber()
        if newCurrentLineNumber != self.currentLineNumber:
            self.currentLineNumber = newCurrentLineNumber
            hi_selection = QTextEdit.ExtraSelection()

hi_selection.format.setBackground(self.currentLineColor)
            hi_selection.format.setProperty(
                QTextFormat.FullWidthSelection, True)
            hi_selection.cursor = self.textCursor()
            hi_selection.cursor.clearSelection()
            self.setExtraSelections([hi_selection])

# Label Code Component
class CodeLabel(QWidget):
    def __init__(self):
        super(CodeLabel, self).__init__()
        layout = QHBoxLayout()
        label1_txt = QLabel()
        label1_txt.setText("<h4>Write or Paste your copy here:
</h4>")
        layout.addWidget(label1_txt)
        self.setLayout(layout)

# Print Label Component
class PrintLabel(QWidget):

    def __init__(self):
        super(PrintLabel, self).__init__()
        vb = QVBoxLayout()
        hb_layout = QHBoxLayout()
        label_text = QLabel()
        plain_text = QPlainTextEdit()

        label_text.setText("Execution <strong> Result Analysis
</strong>")
        label_text.setStyleSheet("color: #2D2D2D;")

        plain_text.setReadOnly(True)
        plain_text.setStyleSheet("background-color: #E5E8ED;")

        hb_layout.addWidget(label_text)
        hb_layout.addStretch(1)

        vb.addLayout(hb_layout)
        vb.addWidget(plain_text)
        self.setLayout(vb)

# Buttons Component
class Buttons(QWidget):

    def __init__(self, editor, print_label):
        layout = QVBoxLayout()
        button_lexer = QPushButton("Run Lexer")
        button_lexer.setFixedSize(100, 40)
        button_lexer.setCursor(QCursor(Qt.PointingHandCursor))
        button_lexer.clicked.connect(lambda:
self.onClickLexer())

        button_parser = QPushButton("Run Parser")
        button_parser.setFixedSize(100, 40)

button_parser.setCursor(QCursor(Qt.PointingHandCursor))
        button_parser.clicked.connect(lambda:
self.onClickParser())

        button_openFile = QPushButton("Open File")
        button_openFile.setFixedSize(100, 40)
button_openFile.setCursor(QCursor(Qt.PointingHandCursor))
        button_openFile.clicked.connect(lambda:
self.openFile(editor))

        layout.addWidget(button_lexer)
        layout.addWidget(button_parser)
        layout.addWidget(button_openFile)
        self.setLayout(layout)

    def onClickLexer(self, editor, print_label):
        tp = print_label.plain_text
        tp.setPlainText("")
        tp.insertPlainText("Lexical Analysis Output\n")
        handleError()
        tokens = runLexerAnalyzer(editor.toPlainText())
        if handleError.lexer_err:
            tp.insertPlainText(
                f"Number of lexer errors:
{handleError.lexer_err}\n")
            tp.insertPlainText(handleError.lexer_err_descript)
        else:
            for tok in tokens:
                tp.insertPlainText("{:4} :
{:4}".format(tok.value, tok.type))
                tp.insertPlainText("\n")
        tp.insertPlainText("\n")
        tp.insertPlainText("\n")

    def onClickedParser(self, editor, print_label):
        tp = print_label.plain_text
        tp.setPlainText("")
        tp.insertPlainText("Syntactic Analysis Output\n")
        handleError()
        tree = runParserAnalyzer(editor.toPlainText())
        if handleError.syntax_err:
            tp.insertPlainText(
                f"Number of syntax errors:
{handleError.syntax_err}\n")

tp.insertPlainText(handleError.syntax_err_descript)
        if handleError.syntax_err:
            tp.insertPlainText(
                f"Number of syntax errors:
{handleError.syntax_err}\n")

tp.insertPlainText(handleError.syntax_err_descript)
        if not handleError.syntax_err and not
handleError.syntax_err:
            tp.insertPlainText("Build Successfully")
            tp.insertPlainText("\n")

    def openFile(self, editor):
        fileSelected = QFileDialog.getOpenFileName()
        path = fileSelected[0]
        print(path)
        with open(path, 'r') as f:
            editor.setPlainText(f.read())

# Main Component
class MainApp(QMainWindow):

    def __init__(self):
        super().__init__()
        self.title = "Dart Analyzer G6"
        self.geometry = (100, 100, 900, 600)
        self.setStyleSheet("background-color: #E5E8ED;")
        self.mountComponents()

    def mountComponents(self):
        self.setWindowTitle(self.title)
        codeLabel = CodeLabel()
        titulo = QLabel()
        titulo.setText("Analizador Dart")
```

```python
        titulo.setStyleSheet("font-size: 16px; font-weight: bold; text-align: center;")
        layout_v1 = QHBoxLayout()
        layout_v2 = QHBoxLayout()
        layout_v1.addWidget(titulo)
        layout_v1.setAlignment(Qt.AlignCenter)
        layout_v2.addWidget(codeLabel)
        main_layout = QVBoxLayout()
        main_layout.addLayout(layout_v1)
        main_layout.addLayout(layout_v2)
        widget = QWidget(self)
        widget.setLayout(main_layout)
        self.setCentralWidget(widget)


if __name__ == "__main__":
    app = QApplication([])
    mainWindow = MainApp()
    mainWindow.show()
    app.exec_()
```