```
from Parser import parseString


def getPythonCode(code):
    '''Convert the given Racecar code into the Python code that will
   run in the GUI.'''

    # first parse the string
    ast = parseString(code)

    # then check for errors
    if len(ast.errors) > 0:
        return (None, ast.errors)

    # then run the string through the semantic analyzer
    # ast = runSemanticAnalyzer(ast)

    # then generate python code!
    pythonCode = generatePythonCode(ast)

    return (pythonCode, None)


def generatePythonCode(ast):
    '''Traverse the AST and output a string containing the python code
   to execute in the GUI.'''

    # potential AST values and their associated translation functions
    # use astTranslators.get() instead of a long chain of else-ifs
    astTranslators = {
        "ID": idTranslator,
        "assignment_command": assignmentCommandTranslator,
        "backward": backwardTranslator,
        "backwards": backwardTranslator,
        "comparison": comparisonTranslator,
        "can_drive_expression": canDriveExpressionTranslator,
        "declaration_command": declarationCommandTranslator,
        "define_command": defineCommandTranslator,
        "drive_command": driveCommandTranslator,
        "empty": emptyTranslator,
        "forward": forwardTranslator,
        "forwards": forwardTranslator,
        "function_command": functionCommandTranslator,
        "getCarPosition": getCarPositionTranslator,
        "if_command": ifCommandTranslator,
        "left": leftTranslator,
        "opt_else": optElseTranslator,
        "opt_else_if": optElseIfTranslator,
        "opt_extra_params": optExtraParamsTranslator,
        "opt_param_list": optParamListTranslator,
        "opt_parameters": optParametersTranslator,
        "plus_expression": plusExpressionTranslator,
        "print": printTranslator,
        "repeat_if_command": repeatIfTranslator,
        "repeat_times_command": repeatTimesTranslator,
        "right": rightTranslator,
        "statement_block": statementBlockTranslator,
        "statements": statementsTranslator,
        "times_expression": timesExpressionTranslator,
        "turn_command": turnCommandTranslator,
        "word_expression": wordExpressionTranslator,
    }

    # "declare" pythonCode since otherwise its first use is inside
```

```
    # an if statement
    pythonCode = ""

    # Fetch the appropriate translator function from astTranslators
    # If there is no translator for ast.value then just let the
    # "translator" be ast.value
    translator = astTranslators.get(ast.value, ast.value)

    # If the "translator" is just a string (inherits from basestring),
    # then return that translator
    if isinstance(translator, basestring):
        pythonCode = ast.value

    # if the translator is a real function then invoke it
    else:
        pythonCode = translator(ast)

    return pythonCode


def indentLines(unindentedLines):
    '''Insert 4 spaces (i.e. 1 tab) at the beginning of every line'''

    splitCode = unindentedLines.splitlines(True)

    pythonCode = "    " + "    ".join(splitCode)
    return pythonCode


def emptyTranslator(ast):
    return ""


def statementsTranslator(ast):
    pythonCode = generatePythonCode(ast.children[0])
    pythonCode += generatePythonCode(ast.children[1])
    return pythonCode


def driveCommandTranslator(ast):
    # drive numSteps direction steps -->
    # translate_car(numSteps, direction)\n
    pythonCode = "translate_car("
    pythonCode += generatePythonCode(ast.children[1])
    pythonCode += ", " + generatePythonCode(ast.children[0])
    pythonCode += ")\n"
    return pythonCode


def forwardTranslator(ast):
    pythonCode = "CarDirection.FORWARDS"
    return pythonCode


def backwardTranslator(ast):
    pythonCode = "CarDirection.BACKWARDS"
    return pythonCode


def turnCommandTranslator(ast):
    pythonCode = "rotate_car("
    pythonCode += generatePythonCode(ast.children[1])
    pythonCode += ")\n"
    return pythonCode

```

```
130.
131. def comparisonTranslator(ast):
132.     pythonCode = generatePythonCode(ast.children[0])
133.     if ast.children[1].value == "is not":
134.         pythonCode += " != "
135.         pythonCode += ast.children[2].value
136.     elif ast.children[1].value == "is":
137.         pythonCode += " == "
138.         pythonCode += generatePythonCode(ast.children[2])
139.     else:
140.         pythonCode += " " + generatePythonCode(ast.children[1])
141.         pythonCode += " " + generatePythonCode(ast.children[2])
142.
143.     return pythonCode
144.
145.
146. def optElseIfTranslator(ast):
147.     pythonCode = "elif "
148.     pythonCode += generatePythonCode(ast.children[1]) + ":\n"
149.     pythonCode += generatePythonCode(ast.children[3])
150.
151.     if ast.children[4].value != "empty":
152.         pythonCode += generatePythonCode(ast.children[4])
153.
154.     return pythonCode
155.
156.
157. def optElseTranslator(ast):
158.     pythonCode = "else:\n"
159.     prelimPythonCode = generatePythonCode(ast.children[2])
160.     pythonCode += generatePythonCode(ast.children[2])
161.     return pythonCode
162.
163.
164. def ifCommandTranslator(ast):
165.     pythonCode = "if " + generatePythonCode(ast.children[1]) + ":\n"
166.     pythonCode += generatePythonCode(ast.children[3])
167.
168.     if ast.children[4].value != "empty":
169.         pythonCode += generatePythonCode(ast.children[4])
170.
171.     if ast.children[5].value != "empty":
172.         pythonCode += generatePythonCode(ast.children[5])
173.     return pythonCode
174.
175.
176. def leftTranslator(ast):
177.     pythonCode = "WheelDirection.LEFT"
178.     return pythonCode
179.
180.
181. def rightTranslator(ast):
182.     pythonCode = "WheelDirection.RIGHT"
183.     return pythonCode
184.
185.
186. def repeatTimesTranslator(ast):
187.     if ast.children[2].value == "times":
188.         pythonCode = "for x in range(" + ast.children[1].value + "):\n"
189.         pythonCode += generatePythonCode(ast.children[4])
190.     return pythonCode
191.
192.
193. def repeatIfTranslator(ast):
194.     pythonCode = "while " + generatePythonCode(ast.children[2]) + ":\n"
```

```
    pythonCode += generatePythonCode(ast.children[4])
    return pythonCode


def declarationCommandTranslator(ast):
    # id is a whatever -->
    # id = None
    pythonCode = generatePythonCode(ast.children[0])
    pythonCode += " = None\n"
    return pythonCode


def idTranslator(ast):
    pythonCode = ast.value
    return pythonCode


def assignmentCommandTranslator(ast):
    pythonCode = generatePythonCode(ast.children[1])
    pythonCode += " = "
    pythonCode += generatePythonCode(ast.children[3])
    pythonCode += "\n"
    return pythonCode


def printTranslator(ast):
    pythonCode = "print_to_console("
    pythonCode += generatePythonCode(ast.children[1])
    pythonCode += ")\n"
    return pythonCode


def defineCommandTranslator(ast):
    pythonCode = "def "
    pythonCode += generatePythonCode(ast.children[0])
    pythonCode += "("
    if ast.children[1].value == "opt_param_list":
        pythonCode += generatePythonCode(ast.children[1])
    pythonCode += "):\n"
    pythonCode += generatePythonCode(ast.children[2])
    return pythonCode


def optParamListTranslator(ast):
    pythonCode = generatePythonCode(ast.children[1])
    if ast.children[5].value == "opt_extra_params":
        pythonCode += generatePythonCode(ast.children[5])
    return pythonCode


def optExtraParamsTranslator(ast):
    pythonCode = ", "
    pythonCode += generatePythonCode(ast.children[1])
    if ast.children[5].value == "opt_extra_params":
        pythonCode += generatePythonCode(ast.children[5])
    return pythonCode


def statementBlockTranslator(ast):
    prelimPythonCode = generatePythonCode(ast.children[0])

    pythonCode = indentLines(prelimPythonCode)

    return pythonCode
```

```
260.
261. def functionCommandTranslator(ast):
262.     pythonCode = generatePythonCode(ast.children[0])
263.     pythonCode += "("
264.     if len(ast.children) > 1:
265.         pythonCode += generatePythonCode(ast.children[1])
266.     pythonCode += ")\n"
267.     return pythonCode
268.
269.
270. def optParametersTranslator(ast):
271.     numChildren = len(ast.children)
272.     if numChildren > 0:
273.         pythonCode = generatePythonCode(ast.children[0])
274.         if numChildren == 2:
275.             pythonCode += ", "
276.             pythonCode += generatePythonCode(ast.children[1])
277.         return pythonCode
278.     else:
279.         return ""
280.
281.
282. def binaryOperatorTranslator(ast):
283.     pythonCode = "(("
284.     pythonCode += generatePythonCode(ast.children[0])
285.     pythonCode += ") "
286.     pythonCode += generatePythonCode(ast.children[1])
287.     pythonCode += " ("
288.     pythonCode += generatePythonCode(ast.children[2])
289.     pythonCode += "))"
290.     return pythonCode
291.
292.
293. def plusExpressionTranslator(ast):
294.     return binaryOperatorTranslator(ast)
295.
296.
297. def timesExpressionTranslator(ast):
298.     return binaryOperatorTranslator(ast)
299.
300. def wordExpressionTranslator(ast):
301.     pythonCode = "(str("
302.     pythonCode += generatePythonCode(ast.children[0])
303.     pythonCode += ") + str("
304.     pythonCode += generatePythonCode(ast.children[2])
305.     pythonCode += "))"
306.     return pythonCode
307.
308. def getCarPositionTranslator(ast):
309.     return "getCurrentPosition()"
310.
311.
312. def canDriveExpressionTranslator(ast):
313.     pythonCode = "can_move("
314.     pythonCode += generatePythonCode(ast.children[1])
315.     pythonCode += ", " + generatePythonCode(ast.children[0])
316.     pythonCode += ")"
317.     return pythonCode
318.
319. if __name__ == "__main__":
320.     inputString = ''
321.     while True:
322.
323.         inputString = raw_input('enter expression > ')
324.
```

```
325.         if inputString == 'exit':
326.             break
327.
328.         else:
329.             print getPythonCode(inputString)
```