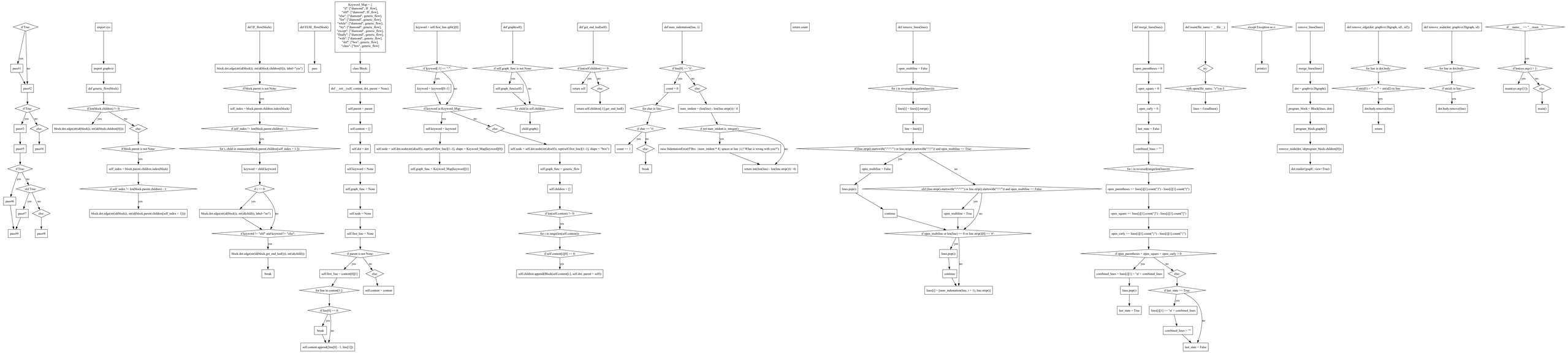
if True:
yes
no
pass#1
pass#2
if True:
yes
no
pass#3
else:
pass#4
pass#5
if True:
yes
no
elif True:
pass#6
yes
no
else:
pass#7
pass#8
pass#9
import sys
import graphviz
def generic_flow(block):
if len(block.children) != 0:
yes
no
block.dot.edge(str(id(block)), str(id(block.children[0])), label="yes")
else:
if block.parent is not None:
yes
if self_index != len(block.parent.children) - 1:
yes
if self_index != len(block.parent.children) - 1:
block.dot.edge(str(id(block)), str(id(block.parent.children[self_index + 1])))
def IF_flow(block):
block.dot.edge(str(id(block)), str(id(block.children[0])), label="yes")
if block.parent is not None:
self_index = block.parent.children.index(block)
for i, child in enumerate(block.parent.children[self_index + 1]):
keyword = child.keyword
if i == 0:
yes
no
block.dot.edge(str(id(block)), str(id(child)), label="no")
if keyword != "elif" and keyword != "else":
yes
block.dot.edge(str(id(block.get_end_leaf())), str(id(child)))
break
def ELSE_flow(block):
pass
Keyword_Map = {
"if": ["diamond", IF_flow],
"elif": ["diamond", IF_flow],
"else": ["diamond", generic_flow],
"for": ["diamond", generic_flow],
"while": ["diamond", generic_flow],
"try": ["diamond", generic_flow],
"except": ["diamond", generic_flow],
"finally": ["diamond", generic_flow],
"with": ["diamond", generic_flow],
"def": ["box", generic_flow],
"class": ["box", generic_flow]
}
class Block:
def __init__(self, content, dot, parent = None):
self.parent = parent
self.content = []
self.dot = dot
self.keyword = None
self.graph_func = None
self.node = None
self.first_line = None
if parent is not None:
yes
no
self.first_line = content[0][1]
else:
self.content = content
for line in content[1:]:
if line[0] == 0:
yes
no
break
self.content.append([line[0] - 1, line[1]])
keyword = self.first_line.split()[0]
if keyword[-1] == ":":
yes
no
keyword = keyword[0:-1]
if keyword in Keyword_Map:
yes
no
else:
self.keyword = keyword
self.node = self.dot.node(str(id(self)), repr(self.first_line[1:-1]), shape = Keyword_Map[keyword][0])
self.graph_func = Keyword_Map[keyword][1]
def graph(self):
if self.graph_func is not None:
yes
no
self.graph_func(self)
for child in self.children:
child.graph()
def get_end_leaf(self):
if len(self.children) == 0:
yes
no
return self
else:
return self.children[-1].get_end_leaf()
def num_indentation(line, i):
if line[i] == "\t":
yes
no
count = 0
else:
for char in line:
if char == "\t":
yes
no
count += 1
else:
break
num_indent = (len(line) - len(line.strip())) / 4
if not num_indent.is_integer():
yes
no
raise IndentationError("Bro. (num_indent * 4) spaces at line {i}? What is wrong with you?")
return int(len(line) - len(line.strip())) / 4
return count
def remove_lines(lines):
open_multiline = False
for i in reversed(range(len(lines))):
lines[i] = lines[i].rstrip()
line = lines[i]
if (line.strip().startswith("""") or line.strip().startswith("""")) and open_multiline == True:
yes
no
open_multiline = False
lines.pop(i)
continue
elif (line.strip().startswith("""") or line.strip().startswith("""")) and open_multiline == False:
yes
no
open_multiline = True
if open_multiline or len(line) == 0 or line.strip()[0] == "#":
yes
no
lines.pop(i)
continue
lines[i] = [num_indentation(line, i + 1), line.strip()]
def merge_lines(lines):
open_parentheses = 0
open_square = 0
open_curly = 0
last_state = False
combined_lines = ""
for i in reversed(range(len(lines))):
open_parentheses += lines[i][1].count("(") - lines[i][1].count(")")
open_square += lines[i][1].count("[") - lines[i][1].count("]")
open_curly += lines[i][1].count("{") - lines[i][1].count("}")
if open_parentheses + open_square + open_curly > 0:
yes
no
combined_lines = lines[i][1] + " " + combined_lines
else:
lines.pop(i)
last_state = True
if last_state == True:
yes
no
lines[i][1] += " " + combined_lines
combined_lines = ""
last_state = False
def main(file_name = "__file__"):
try:
with open(file_name, "r") as f:
lines = f.readlines()
except Exception as e:
print(e)
remove_lines(lines)
merge_lines(lines)
dot = graphviz.Digraph()
program_block = Block(lines, dot)
program_block.graph()
remove_node(dot, id(program_block.children[0]))
dot.render("graph", view=True)
def remove_edge(dot: graphviz.Digraph, id1, id2):
for line in dot.body:
if str(id1) + " -> " + str(id2) in line:
yes
no
dot.body.remove(line)
return
def remove_node(dot: graphviz.Digraph, id):
for line in dot.body:
if str(id) in line:
yes
no
dot.body.remove(line)
if __name__ == "__main__":
if len(sys.argv) > 1:
yes
no
main(sys.argv[1])
else:
main()