ASSIGNMENT 9

```python
# -*- coding: utf-8 -*-
"""Final assignment 9.ipynb

Automatically generated by Colaboratory.

Original file is located at

https://colab.research.google.com/drive/1esXM4qUQxk6CQBsowyuCa1ttMWmcRSYS
"""

keywords = ["LET"
            "PRINT",
            "PRINTLN",
            "INTEGER",
            "INPUT",
            "GOTO",
            "IF"
            "THEN",
            "END"]

operators = ["<", ">", "!", "="]

memory = {}

instructions = {}

variables = {}

variables

contents = ["10 println \" line 10 \" \n",

"20 gosub 200 \n",

"30 println \" line 30 \" \n",

"40 end \n",

"200 println \" in the sub \" \n",

"210 gosub 300 \n",

"220 println \" back from 300 \" \n",

"230 ret \n",

"300 println \" start of 300 sub \" \n",

"310 ret \n",

"320 end \n"
]



contents = list(map(lambda x: x.upper(), contents))

contents
```

```python
input_file = open("input_file.txt", "w")

input_file.writelines(contents)

input_file.close()

input_file = open("input_file.txt", "r")
# print(input_file.read())
input_file.close()

variables

def assign(op : list):
  # print("inside assign")
  # print(op)
  for i in range(1,len(op)):
    if op[i] == "=":
      # assign the value to the variable
      if (op[i+1].isnumeric()):
        # print(op[i+1])
        variables[op[i-1]] = op[i+1]

      # assign the equation to the variable after evaluating the equation
      else:
        # print(op[i+1])
        equation = ""
        for i in range(len(op)-2):
          if op[i+2].isalpha():
            op[i+2] = variables[op[i+2]]
            equation+=str(op[i+2])
            # print("equation: ", equation)
          else:
            equation+=str(op[i+2])


          # assign value to op[0]

          # convert the equation letters to number from dictionary
        # print("after x: ",equation)
        variables[op[0]] = evaluate(equation)

variables

def printf(op : list):
  # print("To be printed: ", op)

  for i in range(len(op)):
    if(op[i].isalpha()):
      # print("inside alpha")
      print(variables[op[i]], end="")
      print()

    else:
      # print("inside quotes")
      if op[i] == "\"":
        string = ""
        i = 1
        while i < len(op)-1:
          string += op[i]
          string += " "
          i+=1
        # for i in range(len(op[:])-2):
        #   string += op[i]
        #   string += " "
        print(string)
        break
      # print(op[i], end="")

def precedence(op):

    if op == '+' or op == '-':
        return 1
    if op == '*' or op == '/':
        return 2
    return 0

def applyOp(a, b, op):

    if op == '+': return a + b
    if op == '-': return a - b
    if op == '*': return a * b
    if op == '/': return a // b

def evaluate(tokens : str) -> int:

    values = []

    ops = []
    i = 0

    while i < len(tokens):

        if tokens[i] == ' ':
            i += 1
            continue

        elif tokens[i] == '(':
            ops.append(tokens[i])

        elif tokens[i].isdigit():
            val = 0
            while (i < len(tokens) and tokens[i].isdigit()):
                val = (val * 10) + int(tokens[i])
                i += 1
            values.append(val)
            i-=1

        elif tokens[i] == ')':
            while len(ops) != 0 and ops[-1] != '(':
                val2 = values.pop()
                val1 = values.pop()
                op = ops.pop()
                values.append(applyOp(val1, val2, op))

            ops.pop()

        else:
            while (len(ops) != 0 and
                precedence(ops[-1]) >=
                   precedence(tokens[i])):

                val2 = values.pop()
                val1 = values.pop()
                op = ops.pop()

                values.append(applyOp(val1, val2, op))

            ops.append(tokens[i])

        i += 1

    while len(ops) != 0:

        val2 = values.pop()
        val1 = values.pop()
        op = ops.pop()

        values.append(applyOp(val1, val2, op))

    return values[-1]




def set_variable(var : str):
  # print(var)
  integers = var.split(" ")
  # print(integers)
  for i in integers:
    variables[i] = 0

  # print("Updated varibales table: ", variables)




def evaluate_condition(condition : str):
  # print(condition)
  left = ""
  right = ""
  i = 0
  while i < len(condition):
    if condition[i] in operators:
      break
    i+=1

  left = condition[:i]
  right = condition[i+1:]
  # print(left)
  # print(right)


  # if len(right) > 1:
  #   right = evaluate(right)

  # if len(left) > 1:
  #   left = evaluate(left)

  if condition[i] == "<":
    ans = int(variables[left]) < int(variables[right])
    return ans
  elif condition[i] == ">":
    ans = int(variables[left]) > int(variables[right])
    return ans

  elif condition[i] == "!":
    ans = int(variables[left]) != int(variables[right])
    return ans

  elif condition[i] == "=":
    ans = int(variables[left]) == int(variables[right])
    return ans
  # equation = str(evaluate(left)) + condition[i] + str(evaluate(right))

def goto(l_number : str):
  # l_number = int(l_number)
  #loop over keys in  memory, find the index of l_number then slice it from
that index to end
  keys = memory.keys()
  req = []
  i = 0
  while(keys[i] != l_number):
    i+=1
  req = memory.keys()[i:]

  print(req)

input_file = open("input_file.txt", "r")

for line in input_file.readlines():

  line_content = line.split(" ")
  print(line_content)

  # split based on line number
  for i in range(len(line_content)):
    memory[line_content[0]] = line_content[1:-1]



  # elif line_content[0] == "PRINT" or line_content[0] == "PRINTLN":
  #   printf(line_content[1:-1])

memory



k = list(memory.keys())
v = list(memory.values())

print(k)

print(v)



line_stack = []
num_stack = []
x = 0
while x < len(v):
  # x+=1
```

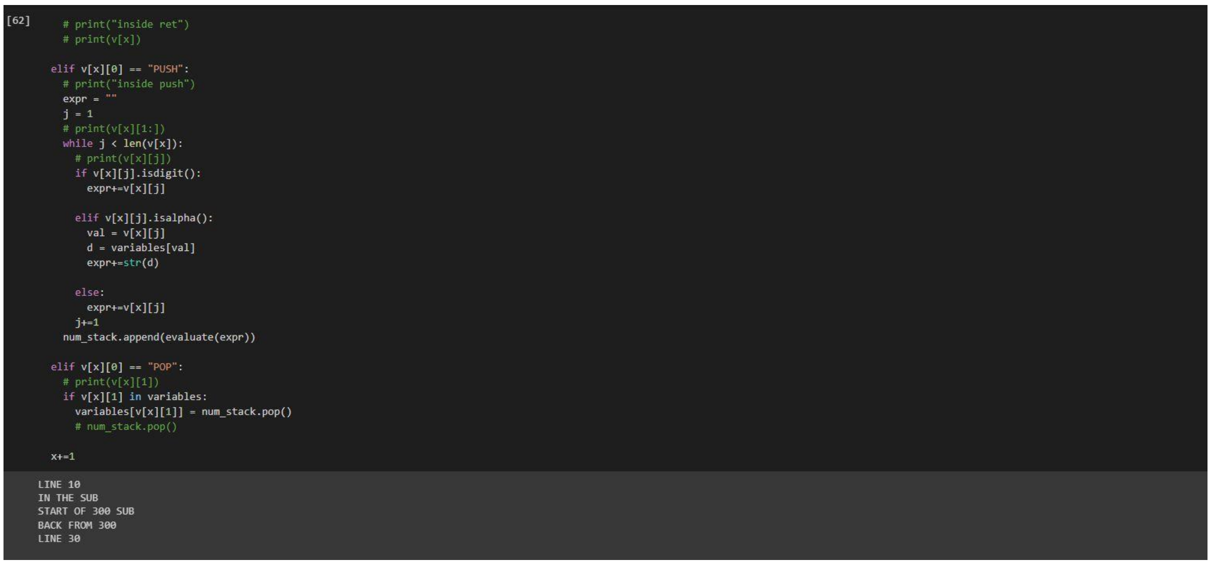
```python
# if memory[line_num][0] == "INTEGER":
  # inst = ""
  # for i in range(1,len(memory[line_num])):
  #   inst += memory[line_num][i]
  # print("inside integer ", inst)
  # inst = inst.replace(",", " ")
  # set_variable(inst)


if v[x][0] == "INTEGER":
  # print("inside integer")
  inst = ""
  for i in range(1,len(v[x])):
    inst += v[x][i]
  # print("inside integer ", inst)
  inst = inst.replace(",", " ")
  set_variable(inst)

elif v[x][0] == "INPUT":
  inst = ""
  # if len(v[x]) > 1:
  #   for i in range(1,len(v[x])):
  #     inst += v[x][i]
  #   inst = inst.split(",")
  #   print("Instruction to input ", inst)
  # else:
  print(v[x])
  if v[x][1] in variables:
    variables[v[x][1]] = input("Enter the value: ")
  else:
    print("Variable doesn't exist")

elif v[x][0] == "LET":
  # inst = ""
  # for i in range(1,len(v[x])):
  assign(v[x][1:])


elif v[x][0] == "END":
  break

elif v[x][0] == "IF":
  condition = ""
  i = 1
  while v[x][i] != "THEN":
    condition += v[x][i]
    i+=1
  # print("If condition: ", condition)

  if evaluate_condition(condition):
    i+=1
    if v[x][i] == "GOTO":
      i+=1
      j=0
      while k[j] != v[x][i]:
        j+=1
      x = j
      # print("index of goto: ", j)
      continue
      # print(v[x])
    elif v[x][i] == "PRINT" or v[x][i] == "PRINTLN":

      printf(v[x][i+1:])
  else:
    x+=1

elif v[x][0] == "PRINT" or v[x][0] == "PRINTLN":
  printf(v[x][1:])

elif v[x][0] == "GOTO":
  j = 0
  while k[j] != v[x][1]:
    j+=1
  x = j

elif v[x][0] == "GOSUB":
  # print("inside go sub")
  line_stack.append(k[x+1])
  # print(line_stack)
  j = 0
  while k[j] != v[x][1]:
    j+=1
  x = j
  continue
  # if (v[x][1] in k):
  #   # x = v[x][1]

  #   print("At ", x, " : ", v[x])

  # else:
  #   print("Line number not found- ", v[x][1])

elif v[x][0] == "RET":
  p = line_stack.pop()
  # print("Popped item: ", p)

  j = 0
  while k[j] != p:
    j+=1

  x = j
  continue
  print("nex v: ", v[x])
  # i = 0
  # while i < len(k):
  #   if k[i] != p:
  #     continue
  #   else:
  #     x = k[i+1]

  # print("inside ret")
  # print(v[x])

elif v[x][0] == "PUSH":
  # print("inside push")
  expr = ""
  j = 1
  # print(v[x][1:])
  while j < len(v[x]):
    # print(v[x][j])
    if v[x][j].isdigit():
```

```python
      expr+=v[x][j]

    elif v[x][j].isalpha():
      val = v[x][j]
      d = variables[val]
      expr+=str(d)

    else:
      expr+=v[x][j]
    j+=1
  num_stack.append(evaluate(expr))

elif v[x][0] == "POP":
  # print(v[x][1])
  if v[x][1] in variables:
    variables[v[x][1]] = num_stack.pop()
    # num_stack.pop()

x+=1
```

INPUT:

```
10 println "line 10"

20 gosub 200

30 println "line 30"

40 end

200 println "in the sub"

210 gosub 300

220 println "back from 300"

230 ret

300 println "start of 300 sub"

310 ret

320 end
```

OUTPUT:



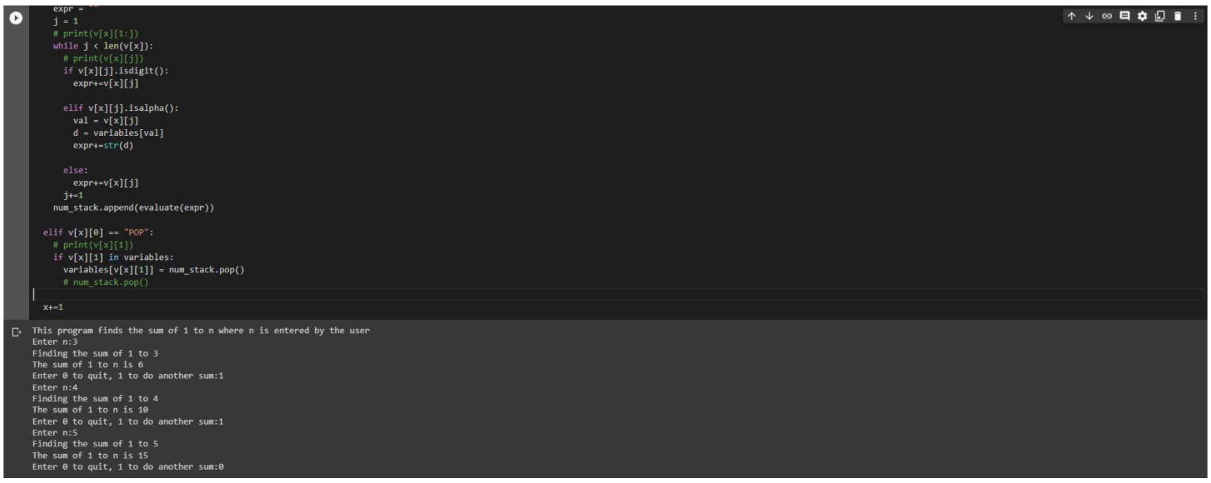
```
[62]     # print("inside ret")
         # print(v[x])

       elif v[x][0] == "PUSH":
         # print("inside push")
         expr = ""
         j = 1
         # print(v[x][1:])
         while j < len(v[x]):
           # print(v[x][j])
           if v[x][j].isdigit():
             expr+=v[x][j]

           elif v[x][j].isalpha():
             val = v[x][j]
             d = variables[val]
             expr+=str(d)

           else:
             expr+=v[x][j]
           j+=1
         num_stack.append(evaluate(expr))

       elif v[x][0] == "POP":
         # print(v[x][1])
         if v[x][1] in variables:
           variables[v[x][1]] = num_stack.pop()
           # num_stack.pop()

       x+=1
```

```
LINE 10
IN THE SUB
START OF 300 SUB
BACK FROM 300
LINE 30
```

INPUT:

```
10 PRINTLN "This program finds the sum of 1 to n where n is entered by the user"

20 INTEGER N, SUM, I

30 PRINT "Enter n:"

40 INPUT n

50 GOSUB 100

60 PRINT "The sum of 1 to n is "

65 PRINTLN SUM

70 PRINT "Enter 0 to quit, 1 to do another sum:"

80 INPUT SUM

90 IF SUM = 1 THEN GOTO 30

95 END

100 PRINTLN "Finding the sum of 1 to ", n

105 LET SUM = 0

110 LET I = 1

120 IF I>N THEN GOTO 160

130 LET SUM = SUM + I

140 LET I = I + 1

150 GOTO 120

160 RET

170 END
```

OUTPUT:



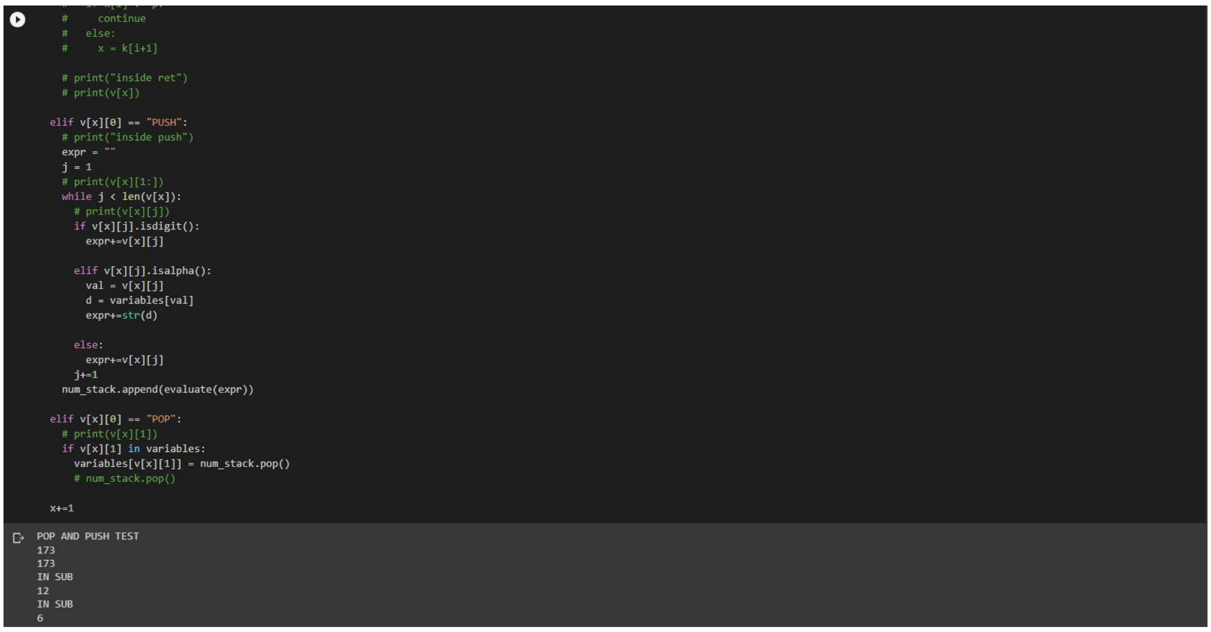
```
    expr = ""
    j = 1
    # print(v[x][1:])
    while j < len(v[x]):
      # print(v[x][j])
      if v[x][j].isdigit():
        expr+=v[x][j]

      elif v[x][j].isalpha():
        val = v[x][j]
        d = variables[val]
        expr+=str(d)

      else:
        expr+=v[x][j]
      j+=1
    num_stack.append(evaluate(expr))

  elif v[x][0] == "POP":
    # print(v[x][1])
    if v[x][1] in variables:
      variables[v[x][1]] = num_stack.pop()
      # num_stack.pop()

  x+=1
```

```
This program finds the sum of 1 to n where n is entered by the user
Enter n:3
Finding the sum of 1 to 3
The sum of 1 to n is 6
Enter 0 to quit, 1 to do another sum:1
Enter n:4
Finding the sum of 1 to 4
The sum of 1 to n is 10
Enter 0 to quit, 1 to do another sum:1
Enter n:5
Finding the sum of 1 to 5
The sum of 1 to n is 15
Enter 0 to quit, 1 to do another sum:0
```

INPUT:

```
5 integer x, y

10 println "pop and push test"

15 integer b

18 let b = 23

20 push 50*3 + b

30 integer a

40 pop a

50 println "a=" , a

60 push a

70 pop b

80 println "b=",b

90 push 5

100 push 7

110 gosub 200

11 push 2

12 push 4

13 gosub 200

120 end

200 println "in sub"

210 pop y

220 pop x

230 println "x+y=", x+y

240 ret

250 end
```

OUTPUT:



```
    #     continue
    #   else:
    #     x = k[i+1]

    # print("inside ret")
    # print(v[x])

  elif v[x][0] == "PUSH":
    # print("inside push")
    expr = ""
    j = 1
    # print(v[x][1:])
    while j < len(v[x]):
      # print(v[x][j])
      if v[x][j].isdigit():
        expr+=v[x][j]

      elif v[x][j].isalpha():
        val = v[x][j]
        d = variables[val]
        expr+=str(d)

      else:
        expr+=v[x][j]
      j+=1
    num_stack.append(evaluate(expr))

  elif v[x][0] == "POP":
    # print(v[x][1])
    if v[x][1] in variables:
      variables[v[x][1]] = num_stack.pop()
      # num_stack.pop()

  x+=1
```

```
POP AND PUSH TEST
173
173
IN SUB
12
IN SUB
6
```

Reference : https://www.geeksforgeeks.org/expression-evaluation/