import math

# funcao principal, que recebe os dados do usuário, verifica se são válidos (positivos, dentro do intervalo, etc)

def menu():

tipo\_fundacao = 0

tipo\_solo = 0

nspt = -1

profundidade\_fundacao = -1

carga\_pilar = -1

while tipo\_fundacao != 1 and tipo\_fundacao != 2:

tipo\_fundacao = int(input("Qual o tipo de fundação? Digite 1 para SAPATA e 2 para ESTACA. \n"))

if tipo\_fundacao != 1 and tipo\_fundacao != 2:

print('Opção inválida. Digite 1 (sapatas) ou 2(estacas).\n')

while tipo\_solo != 1 and tipo\_solo != 2:

tipo\_solo = int(input("Qual o tipo de solo? Digite 1 para areia/silte e 2 para argila. \n"))

if tipo\_solo != 1 and tipo\_solo != 2:

print('Opção inválida. Digite 1 (areia/silte) ou 2(argila).\n')

try:

while nspt < 0:

nspt = int(input("Entre com o número de golpes (N-SPT): \n"))

if nspt < 0:

print('Opção inválida. O número de golpes deve ser positivo.\n')

while profundidade\_fundacao <= 0:

profundidade\_fundacao = float(input("Qual a profundidade da fundação? (Em Metros) \n"))

if profundidade\_fundacao < 0:

print('Valor inválido. A profundidade da fundação deve ser positiva. \n')

while carga\_pilar <= 0:

carga\_pilar = float(input("Qual a carga do pilar? (Em Kg) \n"))

if carga\_pilar < 0:

print('Valor inválido. A carga do pilar deve ser positiva. \n')

except ValueError:

print("Digite os valores como número inteiros.")

# calcula a taxa admissivel do solo dependendo do tipo de solo. Essa funcao recebe dois parâmetros de entrada: um numero inteiro representando o tipo de solo, que pode ser 1 (areia) ou 2 (argila); um numero inteiro que representa o nspt (numero de golpes). Retorna a variavel tx\_solo

def getTaxaAdmSolo(tipo\_solo, nspt):

if tipo\_solo == 1:

tx\_solo = getTaxaAdmSoloAreia(nspt)

elif tipo\_solo == 2:

tx\_solo = getTaxaAdmSoloArgila(nspt)

return tx\_solo

# Essa funcao recebe apenas um parâmetro de entrada: um numero inteiro representando o nspt (numero de golpes) e, a partir desse numero, calcula a taxa admissivel do solo para o tipo Areia/Silte, retornando a variavel tx\_adm\_solo

def getTaxaAdmSoloAreia(nspt):

if 0 <= nspt <= 4:

tx\_adm\_solo = (((nspt - 0) / (4 - 0)) \* (

1 - 0)) + 0 # determina a taxa de solo quando o valor está no intervalo de 0 a 4

print("O valor da taxa de solo é {:.2f} kgf/cm².".format(tx\_adm\_solo))

elif 5 <= nspt <= 8:

tx\_adm\_solo = (((nspt - 5) / (8 - 5)) \* (

2 - 1)) + 1 # determina a taxa de solo quando o valor está no intervalo de 5 a 8

print("O valor da taxa de solo é {:.2f} kgf/cm².".format(tx\_adm\_solo))

elif 9 <= nspt <= 18:

tx\_adm\_solo = (((nspt - 9) / (18 - 9)) \* (

3 - 2)) + 2 # determina a taxa de solo quando o valor está no intervalo de 9 a 18

print("O valor da taxa de solo é {:.2f} kgf/cm².".format(tx\_adm\_solo))

elif nspt >= 19:

tx\_adm\_solo = 4 # quando o valor do ensaio é igual ou maior que 19, a taxa de solo é 4

print("O valor da taxa de solo é {:.2f} kgf/cm².".format(tx\_adm\_solo))

return tx\_adm\_solo

# Essa funcao recebe apenas um parâmetro de entrada: um numero inteiro representando o nspt (numero de golpes) e, a partir desse numero, calcula a taxa admissivel do solo para o tipo Argila, retornando a variavel tx\_adm\_solo

def getTaxaAdmSoloArgila(nspt):

if 0 <= nspt <= 2:

tx\_adm\_solo = (((nspt - 0) / (2 - 0)) \* (

0.25 - 0)) + 0 # determina a taxa de solo quando o valor está no intervalo de 0 a 2

print("O valor da taxa de solo é {:.2f} kgf/cm².".format(tx\_adm\_solo))

elif 3 <= nspt <= 5:

tx\_adm\_solo = (((nspt - 3) / (5 - 3)) \* (

1 - 0.5)) + 0.5 # determina a taxa de solo quando o valor está no intervalo de 3 a 5

print("O valor da taxa de solo é {:.2f} kgf/cm².".format(tx\_adm\_solo))

elif 6 <= nspt <= 10:

tx\_adm\_solo = (((nspt - 6) / (10 - 6)) \* (

3 - 1.5)) + 1.5 # determina a taxa de solo quando o valor está no intervalo de 6 a 10

print("O valor da taxa de solo é {:.2f} kgf/cm².".format(tx\_adm\_solo))

elif 11 <= nspt <= 19:

tx\_adm\_solo = (((nspt - 11) / (19 - 11)) \* (

4 - 3)) + 3 # determina a taxa de solo quando o valor está no intervalo de 11 a 19

print("O valor da taxa de solo é {:.2f} kgf/cm².".format(tx\_adm\_solo))

elif nspt >= 20:

tx\_adm\_solo = 4 # quando o valor do ensaio é igual ou maior que 20, a taxa de solo é 4

print("O valor da taxa de solo é {:.2f} kgf/cm².".format(tx\_adm\_solo))

return tx\_adm\_solo

# funcao que "chama" as funcoes responsaveis por fazer o calculo de cada tipo de fundacao (1 sapata / 2 estaca). Tem como parametros de entrada quatro variaveis: carga proveniente do pilar(float), tipo de solo(int), tipo de fundacao(int) e numero de golpes(int)

def getFundacao(carga\_pilar, tipo\_fundacao, tipo\_solo, nspt):

tx\_adm\_solo = getTaxaAdmSolo(tipo\_solo, nspt)

if tipo\_fundacao == 1:

getDimSapata(carga\_pilar, tx\_adm\_solo)

elif tipo\_fundacao == 2:

getNumEstacas(carga\_pilar)

# funcao responsavel pelo calculo da fundacao do tipo SAPATA, de acordo com o formato (circular 1, quadrada 2, retangular 3). Tem como parametros a carga do pilar e a taxa admissivel do solo, ambaas variaveis do tipo float

def getDimSapata(carga\_pilar, tx\_adm\_solo):

formato = 0

while formato != 1 and formato != 2 and formato != 3:

formato = int(input("\n Indique o formato da sapata (digite 1 para circular, 2 para quadrada e 3 para retangular): \n"))

if formato != 1 and formato != 2 and formato != 3:

print('Opção inválida. O formatos disponíveis são 1 (circular), 2 (quadrada) e 3 (retangular) \n')

# no caso da sapata ser circular, é calculado o valor do diametro ----------------------------------------------------------------------------------------------

elif formato == 1:

raio = (((4 \* carga\_pilar) / (3.14159 \* tx\_adm\_solo)) \*\* 1 / 2) / 2

print("O raio da sapata mede {:.2f} cm.".format(raio))

# no caso da sapata ser quadrada, é calculado o valor do lado da sapata

elif formato == 2:

lado = (carga\_pilar / tx\_adm\_solo) \*\* 1 / 2

print("Os lados da sapata medem {:.2f} cm.".format(lado))

# no caso da sapata ser retangular, é calculado o valor do maior e do menor lado da sapata

elif formato == 3:

a0 = -1

b0 = -1

try:

while a0 <= 0:

a0 = float(input('Qual a medida do maior lado do pilar (em cm)?: '))

if a0 < 0:

print("O valor do maior lado deve ser maior que 0.")

while b0 <= 0:

b0 = float(input('Qual a medida do menor lado do pilar (em cm)?: '))

if b0 < 0:

print("O valor do menor lado deve ser maior que 0.")

if a0 < b0: a0, b0 = b0, a0

# Calculo envolvendo bhaskara

c = a0 - b0

c\_a = 1

c\_b = c

c\_c = - (carga\_pilar / tx\_adm\_solo)

b1 = (- c\_b + ((c\_b \*\* 2) - 4 \* c\_a \* c\_c) \*\* (1 / 2)) / (2 \* c\_a)

b2 = (- c\_b - ((c\_b \*\* 2) - 4 \* c\_a \* c\_c) \*\* (1 / 2)) / (2 \* c\_a)

if b2 < 0:

b = b1

else:

print("Erro")

a = c + b

# Fim da baskara

print("O lado maior da sapata mede {:.2f} cm e o lado menor {:.2f} cm.".format(a, b))

except ValueError:

print("Digite os valores como número inteiros.")

# de acordo com o tipo, é calculado o numero necessario de estacas. Só tem um parametro de entrada: carga proveniente do pilar

def getNumEstacas(carga\_pilar):

tipo\_fundacao = int(

input("\n Indique o tipo de estaca (digite 1 para Strauss, 2 para Franki e 3 para Broca): \n"))

if tipo\_fundacao == 1:

getStrauss()

elif tipo\_fundacao == 2:

getFranki()

elif tipo\_fundacao == 3:

getBroca()

quant\_estacas = math.ceil((carga\_pilar / carga\_estaca))

print("A quantidade de estacas é", quant\_estacas, "de", diametro, "cm.")

# funcao responsavel por converter o valor do diametro informado pelo usuario, de cm para kgf, para o tipo de estaca Strauss \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

def getStrauss():

global diametro

diametro = 0

global carga\_estaca

lista\_strauss = [200, 300, 450, 600, 800]

d25, d32, d38, d45, d55 = lista\_strauss

try:

while diametro != 25 and diametro != 32 and diametro != 38 and diametro != 45 and diametro != 55:

diametro = float(input("Informe o valor do diametro: "))

if diametro != 25 and diametro != 32 and diametro != 38 and diametro != 45 and diametro != 55:

print(

'Opção inválida. Informe valores de diâmetro correspondentes aos disponíveis, de acordo com a tabela ao lado.\n')

elif diametro == 25:

carga\_estaca = d25

elif diametro == 32:

carga\_estaca = d32

elif diametro == 38:

carga\_estaca = d38

elif diametro == 45:

carga\_estaca = d45

elif diametro == 55:

carga\_estaca = d55

except ValueError:

print("Digite os valores como número inteiros.")

return carga\_estaca

# funcao responsavel por converter o valor do diametro informado pelo usuario, de cm para kgf, para o tipo de estaca Franki \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

def getFranki():

global diametro

diametro = 0

global carga\_estaca

lista\_franki = [550, 750, 1300, 1700]

d35, d40, d52, d60 = lista\_franki

try:

while diametro != 35 and diametro != 40 and diametro != 52 and diametro != 60:

diametro = float(input("Informe o valor do diametro: "))

if diametro != 35 and diametro != 40 and diametro != 52 and diametro != 60:

print(

'Opção inválida. Informe valores de diâmetro correspondentes aos disponíveis, de acordo com a tabela ao lado.\n')

elif diametro == 35:

carga\_estaca = d35

elif diametro == 40:

carga\_estaca = d40

elif diametro == 52:

carga\_estaca = d52

elif diametro == 60:

carga\_estaca = d60

except ValueError:

print("Digite os valores como número inteiros.")

return carga\_estaca

# funcao responsavel por converter o valor do diametro informado pelo usuario, de cm para kgf, para o tipo de estaca Broca \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

def getBroca():

global diametro

diametro = 0

global carga\_estaca

lista\_broca = [50, 70, 100]

d20, d25, d32 = lista\_broca

try:

while diametro != 20 and diametro != 25 and diametro != 32:

diametro = float(input("Informe o valor do diametro: "))

if diametro != 20 and diametro != 25 and diametro != 32:

print(

'Opção inválida. Informe valores de diâmetro correspondentes aos disponíveis, de acordo com a tabela ao lado.\n')

elif diametro == 20:

carga\_estaca = d20

elif diametro == 25:

carga\_estaca = d25

elif diametro == 32:

carga\_estaca = d32

except ValueError:

print("Digite os valores como número inteiros.")

return carga\_estaca

getFundacao(carga\_pilar, tipo\_fundacao, tipo\_solo, nspt)

if \_\_name\_\_ == '\_\_main\_\_':

menu()