№5

import math  
  
  
def main(y, x):  
 n = len(y)  
 y.insert(0, 0)  
 x.insert(0, 0)  
 total = 0  
 for i in range(1, n + 1):  
 left = 92 \* y[n+1-i]  
 right = x[n+1-i] \*\* 2  
 total += (left - right)  
 return 67 \* total

№6

from itertools import permutations

def zero(items, left, right):

if items[0] == 'FLUX':

return left

if items[0] == 'NSIS':

return right

def one(items, left, right):

if items[1] == 2019:

return left

if items[1] == 1968:

return right

def two(items, left, middle, right):

if items[2] == 1984:

return left

if items[2] == 1968:

return middle

if items[2] == 1989:

return right

def three(items, left, middle, right):

if items[1] == 'VCL':

return left

if items[1] == 'XPROC':

return middle

if items[1] == 'KIT':

return right

def func(items):

return zero(

items,

two(

items,

one(items, 0, 1),

2,

3

),

three(items, 4, two(items, 5, 6, 7), two(items, 8, 9, 10))

)

def main(items):

comb = list(permutations(items))

i = 0

x = None

while x is None:

x = func(comb[i])

i += 1

return x

def main(x):  
 if x[0] == "NSIS":  
 return three(x)  
 else:  
 if x[2] == 1984:  
 if x[1] == 2019:  
 return 0  
 else:  
 return 1  
 if x[2] == 1968:  
 return 2  
 else:  
 return 3  
  
  
def three(x):  
 if x[3] == 'XPROC':  
 if x[2] == 1984:  
 return 5  
 if x[2] == 1968:  
 return 6  
 else:  
 return 7  
 if x[3] == 'KIT':  
 if x[2] == 1984:  
 return 8  
 if x[2] == 1968:  
 return 9  
 else:  
 return 10  
 else:  
 return 4

№7

def main(input):  
 a = (input & 0b00000000000000000000011111111111) << 16  
 b = (input & 0b00000000001111111111100000000000) >> 6  
 c = (input & 0b00000011110000000000000000000000) >> 21  
 d = (input & 0b01111100000000000000000000000000) << 1  
 e = (input & 0b10000000000000000000000000000000) >> 31  
 return a | b | c | d | e

№8

def main(txt):  
 result = {}  
 mass = txt.split(',')[:-1]  
 for x in mass:  
 val, per = x.split('>')  
 val = val.split()[-2]  
 val = int(val)  
 per = per[1:-2].strip().strip('`')  
 result[per] = val  
 return result

№9

from enum import Enum  
  
  
class State(Enum):  
 A = 0  
 B = 1  
 C = 2  
 D = 3  
 E = 4  
 F = 5  
 G = 6  
  
  
class StateMachine:  
 state = State.A  
  
 def fork(self):  
 return self.update({  
 State.A: [State.B, 0],  
 State.E: [State.F, 7],  
 })  
  
 def coat(self):  
 return self.update({  
 State.B: [State.C, 2],  
 State.D: [State.B, 6],  
 State.C: [State.D, 4],  
 State.A: [State.D, 1],  
 State.F: [State.G, 8],  
 })  
  
 def build(self):  
 return self.update({  
 State.B: [State.F, 3],  
 State.D: [State.E, 5],  
 State.F: [State.F, 9],  
 })  
  
 def update(self, transitions):  
 self.state, signal = transitions[self.state]  
 return signal  
  
  
def main():  
 return StateMachine()

№10

def delete\_empty\_rows(table):  
 return [row for row in table if row[0] is not None]  
  
  
def transpose(table):  
 response = []  
 for i in range(len(table[0])):  
 response.append([])  
 for j in range(len(table)):  
 response[i].append(table[j][i])  
 return response  
  
  
def delete\_duplicate\_columns(table):  
 x = []  
 for row in table:  
 if row in x:  
 continue  
 x.append(row)  
 table = x  
 for row in table:  
 del row[3]  
 return table  
  
  
x = 0  
  
  
def transformer(i, value):  
 global x  
 if i == 0:  
 return value[5:]  
 if i == 1:  
 day, month, year = value.split('/')  
 return f'{year}/{month}/{day}'  
 if i == 2:  
 f, y = value.split('|')  
 f = (float(f[2:6]))  
 f = f/100  
 f = round(f)  
 f = str(f) + '%'  
 y = y[5:] + ' ' + y[:2]  
 return y, f  
 if i == 3:  
 print(x)  
 return x  
  
  
def transform(table):  
 for i in range(len(table)):  
 for j in range(len(table[i])):  
 table[i][j] = transformer(i, table[i][j])  
 return table  
  
  
def fix(table):  
 for row in table:  
 k = row[2]  
 del row[2]  
 row.append(k[0])  
 row.append(k[1])  
 return table  
  
  
def main(table):  
 return fix(transpose(  
 transform(  
 transpose(  
 delete\_duplicate\_columns(  
 delete\_empty\_rows(table)  
 )  
 )  
 )  
 )  
 )

№1

import math  
  
  
def main(y, x):  
 a = math.sqrt(((x \*\* 2) / 73 - 13 \* y \*\* 3 - 1) \*\* 4 + 91)  
 b = 32 \* ((x \*\* 2) / 86 - x) + 81 \* (y + 1 + 70 \* x \*\* 3) \*\* 6  
 c = 59 \* y \*\* 3 + math.sin(52 \* x - 72 \* x \*\* 3 - x \*\* 2 / 71) \*\* 5  
 return a + b / c

№2

import math  
  
  
def main(y):  
 if y < -55:  
 return (64 \* y \*\* 6 - y \*\* 4/13 - y \*\* 5)  
 if -55 <= y < 27:  
 return 82 \* (y-89 \* y \*\* 2) \*\* 6 + (math.fabs(y) \*\* 7) +\  
 (math.ceil(93 - y \*\* 3)) \*\* 4  
 if 27 <= y < 123:  
 return (54 \* y) \*\* 6 - 5 \* y  
 if y >= 123:  
 return math.fabs(y) \*\* 3  
k = main(123)  
print (k)

№3

import math  
  
  
def main(a, y):  
 total = 0  
 for i in range(1, a+1):  
 left = (36 \* (i \*\* 3) + y)  
 right = 11 \* (68 - i \*\* 3) \*\* 2  
 mid = left - right  
 total += mid  
 return total

№4

def main(n):  
 if n == 0:  
 return -0.65  
 if n == 1:  
 return 0.03  
 if n >= 2:  
 a = 12 \* (main(n - 1) \*\* 2) + main(n - 2)  
 b = 17 \* (main(n - 1) \*\* 3)  
 return (a+b)/34  
k = main(2)  
print(k)

№11

from struct import \*  
  
FMT = dict(  
 char='c',  
 int8='b',  
 uint8='B',  
 int16='h',  
 uint16='H',  
 int32='i',  
 uint32='I',  
 int64='q',  
 uint64='Q',  
 float='f',  
 double='d'  
)  
  
  
def parse(buf, offs, ty):  
 return unpack\_from(FMT[ty], buf, offs)[0], offs + calcsize(FMT[ty])  
  
  
def parse\_a(buf, offs):  
 a1 = []  
 for \_ in range(4):  
 val, offs = parse(buf, offs, 'char')  
 a1.append(val.decode())  
 a2, offs = parse(buf, offs, 'double')  
 a3, offs = parse(buf, offs, 'int32')  
 a4, offs = parse(buf, offs, 'double')  
 a5, offs = parse(buf, offs, 'uint16')  
 a6, offs = parse(buf, offs, 'int32')  
 a7 = []  
 for \_ in range(5):  
 val, offs = parse\_b(buf, offs)  
 a7.append(val)  
 a8, offs = parse\_c(buf, offs)  
 return dict(A1=''.join(a1), A2=a2, A3=a3, A4=a4, A5=a5, A6=a6, A7=a7, A8=a8), offs  
  
  
def parse\_b(buf, offs):  
 b1, offs = parse(buf, offs, 'int16')  
 b2, offs = parse(buf, offs, 'uint32')  
 return dict(B1=b1, B2=b2), offs  
  
  
def parse\_c(buf, offs):  
 c1, offs = parse(buf, offs, 'uint16')  
 c2\_offs, offs = parse(buf, offs, 'uint16')  
 c2, c2\_offs = parse\_d(buf, c2\_offs)  
 c3, offs = parse(buf, offs, 'int8')  
 c4, offs = parse(buf, offs, 'double')  
 return dict(C1=c1, C2=c2, C3=c3, C4=c4), offs  
  
  
def parse\_d(buf, offs):  
 d1 = []  
 for \_ in range(8):  
 val, offs = parse(buf, offs, 'uint16')  
 d1.append(val)  
 d2, offs = parse(buf, offs, 'int16')  
 d3, offs = parse(buf, offs, 'float')  
 d4 = []  
 for \_ in range(7):  
 val, offs = parse(buf, offs, 'uint8')  
 d4.append(val)  
 return dict(D1=d1, D2=d2, D3=d3, D4=d4), offs  
  
  
def main(buf):  
 return parse\_a(buf, 5)[0]

от младшего к старшему – обычный словарь

от старшего к младшему - >Q,>q