```python
import json
import re

def normalize_string(value):
    if not value:
        return ""
    if not isinstance(value, str):
        value = str(value)
    value.lower().strip().replace(" ", "").replace("-", "").replace("_",
"").replace("\n", " ").replace("\t", "")
    return re.sub(r"\s+", " ", value)

def search_key(data, target_key, default_value=''):
    target_key = normalize_string(target_key)
    stack = [data]

    while stack:
        current = stack.pop(0)
        if isinstance(current, dict):
            for key, value in current.items():
                normalized_key = normalize_string(key)
                if target_key == normalized_key:
                    return value
                stack.append(value)

        elif isinstance(current, list):
            stack.extend(current)
    return 0 if isinstance(target_key, int) else default_value

def search_second_key(data, target_key, default_value=''):
    target_key = normalize_string(target_key)
    stack = [data]
    matches = []

    while stack:
        current = stack.pop(0)
        if isinstance(current, dict):
            for key, value in current.items():
                if normalize_string(key) == target_key:
                    matches.append(value)
                stack.append(value)
        elif isinstance(current, list):
            stack.extend(current)
    return matches[1] if len(matches) > 1 else default_value

def diagram_number_pdf(data, key_variants):
    if isinstance(data, dict):
        for key, value in data.items():
            if normalize_string(key) in [normalize_string(k) for k in key_variants]:
                return value

            deeper_result = diagram_number_pdf(value, key_variants)
            if deeper_result is not None:
                return deeper_result

    elif isinstance(data, list):
        for item in data:
            deeper_result = diagram_number_pdf(item, key_variants)
            if deeper_result is not None:
                return deeper_result
    return None

def extract_float_value(value):
    if isinstance(value, str):
        numeric_part = ''.join(c if c.isdigit() or c == '.' else '' for c in value)
        try:
            return float(numeric_part) if numeric_part else 0.0
        except ValueError:
            return 0.0
    elif isinstance(value, (int, float)):
        return float(value)
    return 0.0


# Files paths
# ------------------------------------------------------------------------------------
--------------------------------------------------
with open(r"/Users/umerjillani/Documents/Upwork/EC/Milestone3/Code/31.json") as f:
    data_pdf = json.load(f)

with open(r"/Users/umerjillani/Documents/Upwork/EC/Milestone3/Code/application.json")
as f:
    data_app = json.load(f)
# ------------------------------------------------------------------------------------
---------------------------------------------------


city_value_pdf = search_key(data_pdf, 'City')
state_value_pdf = search_key(data_pdf, 'State')
zipcode_value_pdf = search_key(data_pdf, 'ZIPCode')

city_value_app = search_key(data_app, 'city')
state_value_app = search_key(data_app, 'state')
zipcode_value_app = search_key(data_app, 'zipCode')

print("Rule 1\n----------------------------------------------------")
print(f"City from PDF: {city_value_pdf}")
```

```python
print(f"State from PDF: {state_value_pdf}")
print(f"ZIPCode from PDF: {zipcode_value_pdf}")

print(f"\nCity from Application: {city_value_app}")
print(f"State from Application: {state_value_app}")
print(f"ZIPCode from Application: {zipcode_value_app}")

# Compare the normalized values
if (city_value_pdf == city_value_app or state_value_pdf == state_value_app) and
zipcode_value_pdf == zipcode_value_app:
    print("Parameters matched.\n")
else:
    print("Parameters do not match.\n")

# Diagram Numbers
print("Rule 2\n----------------------------------------------------")

key_variants = [
    "BuildingDiagramNumber",
    "Building Diagram Number",
    "building_diagram_number",
    "bldg_diag_num",
    "buildingDiagramNo",
    "Diagram Number"
]

try:
    diagramNumber_pdf = diagram_number_pdf(data_pdf, key_variants)
except (ValueError, TypeError):
    diagramNumber_pdf = -1

try:
    diagram_number_app = search_key(data_app, "ecDiagramNumber")
except (ValueError, TypeError):
    diagram_number_app = -1

print(f"PDF diagram number : {diagramNumber_pdf}\nApplication diagram number :
{diagram_number_app}\n")

if diagramNumber_pdf and diagram_number_app:
    if str(diagramNumber_pdf).strip() == str(diagram_number_app).strip():
        print("Diagram Numbers matched")
    else:
        print("Diagram Numbers don't match.")
else:
    print("One of the diagram numbers was not found.")
```

```
# Crawlspace and Garage Square Footage details
print("\nRule 3\n----------------------------------------------------")

crawlspace_details = search_key(data_pdf, 'CrawlspaceDetails')
if crawlspace_details and 'SquareFootage' in crawlspace_details:
    crawlspace_square_footage =
extract_float_value(crawlspace_details['SquareFootage'])
else:
    crawlspace_square_footage = 0

garage_details = search_key(data_pdf, 'GarageDetails')
if garage_details and 'SquareFootage' in garage_details:
    garage_square_footage = extract_float_value(garage_details['SquareFootage'])
else:
    garage_square_footage = 0

enclosure_Size = search_key(data_app, "enclosureSize")
if isinstance(enclosure_Size, dict):
    enclosure_Size = enclosure_Size.get("enclosureSize", 0)

total_square_footage = crawlspace_square_footage + garage_square_footage
diagram_no_choices = ['6', '7', '8', '9']

if diagramNumber_pdf in diagram_no_choices:

    if (total_square_footage == 0 and enclosure_Size is None) or (total_square_footage
== enclosure_Size):
        print(f"Diagram Number is among {', '.join(map(str, diagram_no_choices))}. and
Crawlspace and Garage square footage '{crawlspace_square_footage +
garage_square_footage}' is aligned with Total Enclosure size '{enclosure_Size}' in the
application.")
    else:
        print(f"Diagram Number is among {', '.join(map(str, diagram_no_choices))}. and
Crawlspace and Garage square footage '{crawlspace_square_footage +
garage_square_footage}' is not aligned with Total Enclosure size '{enclosure_Size}' in
the application.")
else:
    print(f"Diagram Number is not among {', '.join(map(str, diagram_no_choices))}. so
no comparison is required.")

# Searching for CBRS/OPA details  cbrsSystemUnitOrOpa
print("\nRule 4\n----------------------------------------------------")
CBRS = search_key(data_pdf, 'CBRS')
OPA = search_key(data_pdf, 'OPA')
CBRS_OPA_app = search_key(data_app, 'cbrsSystemUnitOrOpa')

if CBRS_OPA_app != CBRS or CBRS_OPA_app != OPA:
    print("CBRS/OPA details do not match.")
```

```
else:
    print("CBRS/OPA details matched with the application.")

if CBRS.lower() == "yes" or OPA.lower() == "yes":
    print("Area in CBRS/OPA, Additional Documentation Required.")
else:
    print("Area not in CBRS/OPA, Additional Documentation Not Required.\n")

# Rule 5
Construction_status_pdf_V1 = search_key(data_pdf, 'Building_Elevations_Based_On')
Construction_status_pdf_V2 = search_key(data_pdf, 'Elevation_Basis')
Construction_status_pdf_V3 = search_key(data_pdf, 'BuildingUnderConstruction')
Construction_status_app = search_key(data_app, 'buildingUnderConstruction')

print("Rule 5\n----------------------------------------------------")
if normalize_string(Construction_status_pdf_V1) == "finishedconstruction" and
Construction_status_app:
    print("Construction status PDF: Finished Construction.\nConstruction Status
Application: Finished Construction.\n")

elif normalize_string(Construction_status_pdf_V1) == "finishedconstruction" and not
Construction_status_app:
    print("Construction status PDF: Finished Construction.\nConstruction Status
Application: Under Construction.\n")

elif normalize_string(Construction_status_pdf_V2) == "finishedconstruction" and
Construction_status_app:
    print("Construction status PDF: Finished Construction.\nConstruction Status
Application: Finished Construction.\n")

elif normalize_string(Construction_status_pdf_V2) == "finishedconstruction" and not
Construction_status_app:
    print("Construction status PDF: Finished Construction.\nConstruction Status
Application: Under Construction.\n")

elif normalize_string(Construction_status_pdf_V3) == "finishedconstruction" and
Construction_status_app:
    print("Construction status PDF: Finished Construction.\nConstruction Status
Application: Finished Construction.\n")

elif normalize_string(Construction_status_pdf_V3) == "finishedconstruction" and not
Construction_status_app:
    print("Construction status PDF: Finished Construction.\nConstruction Status
Application: Under Construction.\n")

elif normalize_string(Construction_status_pdf_V1) != "finishedconstruction" and
Construction_status_app:
    print("Construction status PDF: Under Construction.\nConstruction Status
Application: Finished Construction.\n")

elif normalize_string(Construction_status_pdf_V1) != "finishedconstruction" and not
Construction_status_app:
    print("Construction status PDF: Under Construction.\nConstruction Status
Application: Under Construction.\n")

elif normalize_string(Construction_status_pdf_V2) != "finishedconstruction" and
Construction_status_app:
    print("Construction status PDF: Under Construction.\nConstruction Status
Application: Finished Construction.\n")

elif normalize_string(Construction_status_pdf_V2) != "finishedconstruction" and not
Construction_status_app:
    print("Construction status PDF: Under Construction.\nConstruction Status
Application: Under Construction.\n")

elif normalize_string(Construction_status_pdf_V3) != "finishedconstruction" and
Construction_status_app:
    print("Construction status PDF: Under Construction.\nConstruction Status
Application: Finished Construction.\n")

elif normalize_string(Construction_status_pdf_V3) != "finishedconstruction" and not
Construction_status_app:
    print("Construction status PDF: Finished Construction.\nConstruction Status
Application: Under Construction.\n")


# Rule 6
def validate_signature_requirements(data_pdf):

    print("\nRule 6 - Signature Validation\n------------------------------------------
----------")



    # Check if Section C measurements are used

    section_c_measurements_used = any([

        extract_float_value(search_key(data_pdf, 'Section C', {}).get('Top of Bottom
Floor')),

        extract_float_value(search_key(data_pdf, 'Section C', {}).get('Lowest Adjacent
Grade (LAG)')),
        extract_float_value(search_key(data_pdf, 'Section C', {}).get('Lowest Floor
Elevation'))

    ])



    # Check if Section E measurements are used

    section_e_measurements_used = extract_float_value(search_key(data_pdf, 'Section
E', {}).get('Top of Bottom Floor')) > 0



    # Check if Section H measurements are used

    section_h_measurements_used = extract_float_value(search_key(data_pdf, 'Section
H', {}).get('Top of Bottom Floor')) > 0



    # Extracting relevant information from required sections

    section_d_signature = search_key(data_pdf, 'Section D', {}).get('Surveyor
Signature')

    section_d_license = search_key(data_pdf, 'Section D', {}).get('Surveyor License
Number')

    section_d_name = search_key(data_pdf, 'Section D', {}).get('Surveyor Name')



    section_f_signature = search_key(data_pdf, 'Section F', {}).get('Representative
Signature')

    section_f_name = search_key(data_pdf, 'Section F', {}).get('Representative Name')



    section_i_signature = search_key(data_pdf, 'Section I', {}).get('Representative
Signature')

    section_i_name = search_key(data_pdf, 'Section I', {}).get('Representative Name')



    # Validation checks

    if section_c_measurements_used:
        if section_d_signature and section_d_license and section_d_name:

            print("Section C measurements are used, and Section D includes Surveyor's 
Name, License #, and Signature.")

        else:

            print(" Section C measurements are used, but Section D is missing required 
details.")



    if section_e_measurements_used:

        if section_f_signature and section_f_name:

            print("Section E measurements are used, and Section F includes 
Representative's Name and Signature.")

        else:

            print("Section E measurements are used, but Section F is missing required 
details.")



    if section_h_measurements_used:

        if section_i_signature and section_i_name:

            print("Section H measurements are used, and Section I includes 
Representative's Name and Signature.")

        else:

            print("Section H measurements are used, but Section I is missing required 
details.")



    # Final decision

    if not (section_c_measurements_used or section_e_measurements_used or 
section_h_measurements_used):

        print("No Section C, E, or H measurements were used, so signature validation 
is not required.")
```

```python
# Rule 7 - Elevation Logic
#-----------------------------------------------------------------------------------
print("Rule 7\n----------------------------------------------------")

try:
    Section_C_FirstFloor_Height_app = float(search_key(data_app,
'ecSectionCFirstFloorHeight'))
except (ValueError, TypeError):
    Section_C_FirstFloor_Height_app = 0

try:
    Section_C_LAG_app = float(search_key(data_app, 'ecSectionCLowestAdjacentGrade'))
except (ValueError, TypeError):
    Section_C_LAG_app = 0

try:
    Section_C_Lowest_Floor_Elevation_app = float(search_key(data_app,
'ecSectionCLowestFloorElevation'))
except (ValueError, TypeError):
    Section_C_Lowest_Floor_Elevation_app = 0

section_c_measurements_used = False

if Section_C_FirstFloor_Height_app > 0 or Section_C_LAG_app > 0 or
Section_C_Lowest_Floor_Elevation_app > 0:
    section_c_measurements_used = True
    print("\nA: Section C measurements are used in the application.\n")
else:
    print("\nA: Section C measurements are not used in the application.\n")

section_C = search_key(data_pdf, 'Section C')
top_of_bottom_floor_pdf = extract_float_value(search_key(section_C, 'Top of Bottom
Floor'))
top_of_next_higher_floor_pdf = extract_float_value(search_key(section_C, 'Top of Next
Higher Floor'))
LAG_pdf = extract_float_value(search_key(section_C, 'Lowest Adjacent Grade (LAG)'))
HAG_pdf = extract_float_value(search_key(section_C, 'Highest Adjacent Grade'))

diagram_choices_1 = ['1', '1a', '3', '6', '7', '8']
diagram_choices_2 = '1b'
diagram_choices_3 = ['2', '2a', '2b', '4', '9']
diagram_choices_4 = '5'
diagram_choices_5 = ['2', '4', '6', '7', '8', '9']
```

```python
if section_c_measurements_used:
    if top_of_bottom_floor_pdf == Section_C_FirstFloor_Height_app:
        print("EC Top of Bottom Floor matches with Section C First Floor Height in Application.\n")
    else:
        print("Red Flag -> EC Top of Bottom Floor does not matche with Section C First Floor Height in Application.\n")

    if LAG_pdf == Section_C_LAG_app:
        print("EC Lowest Adjacent Grade (LAG) matches with Section C LAG in Application.\n")
    else:
        print("Red Flag -> EC Lowest Adjacent Grade (LAG) does not match with Section C LAG in Application.\n")

    if diagramNumber_pdf.lower() in diagram_choices_1:
        if LAG_pdf <= top_of_bottom_floor_pdf <= LAG_pdf + 2:
            print(f"C2a: For diagram no. in '{', '.join(map(str, diagram_choices_1))}' The top of bottom floor elevation should be within 2 feet of the LAG, but not below the LAG.\nTop of Bottom Floor: {top_of_bottom_floor_pdf}\nLAG: {LAG_pdf}\nElevation Logic matched.\n")
        else:
            print(f"C2a: For diagram no. in '{', '.join(map(str, diagram_choices_1))}' The top of bottom floor elevation should be within 2 feet of the LAG, but not below the LAG.\nTop of Bottom Floor: {top_of_bottom_floor_pdf}\nLAG: {LAG_pdf}\nRed Flag -> Elevation Logic not matched.\n")

    elif diagramNumber_pdf.lower() == diagram_choices_2:
        if LAG_pdf <= top_of_bottom_floor_pdf <= LAG_pdf + 6:
            print(f"C2a: For diagram no. in '{diagram_choices_2}' The elevation should be within 6 feet of the LAG, but not below the LAG.\nTop of Bottom Floor: {top_of_bottom_floor_pdf}\nLAG: {LAG_pdf}\nElevation Logic matched.\n")
        else:
            print(f"C2a: For diagram no. in '{diagram_choices_2}' The elevation should be within 6 feet of the LAG, but not below the LAG.\nTop of Bottom Floor: {top_of_bottom_floor_pdf}\nLAG: {LAG_pdf}\nRed Flag -> Elevation Logic not matched.\n")

    elif diagramNumber_pdf.lower() in diagram_choices_3:
        if top_of_bottom_floor_pdf < LAG_pdf:
            print(f"C2a: For diagram no. in '{', '.join(map(str, diagram_choices_3))}' The elevation should be below the LAG.\nTop of Bottom Floor: {top_of_bottom_floor_pdf}\nLAG: {LAG_pdf}\nElevation Logic matched.\n")
        else:
            print(f"C2a: For diagram no. in '{', '.join(map(str, diagram_choices_3))}' The elevation should be below the LAG.\nTop of Bottom Floor: {top_of_bottom_floor_pdf}\nLAG: {LAG_pdf}\nRed Flag -> Elevation Logic not matched.\n")
    elif diagramNumber_pdf.lower() in diagram_choices_4:
        if LAG_pdf <= top_of_bottom_floor_pdf <= (LAG_pdf + 20):
            print(f"C2a: For diagram no. in '{diagram_choices_4}' The elevation should
be within 20 feet of the LAG, but not below the LAG.\nTop of Bottom Floor:
{top_of_bottom_floor_pdf}\nLAG: {LAG_pdf}\nElevation Logic matched.\n")
        else:
            print(f"C2a: For diagram no. in '{diagram_choices_4}' The elevation should
be within 20 feet of the LAG, but not below the LAG.\nTop of Bottom Floor:
{top_of_bottom_floor_pdf}\nLAG: {LAG_pdf}\nRed Flag -> Elevation Logic not
matched.\n")

    if diagramNumber_pdf.lower() in diagram_choices_5:
        if top_of_next_higher_floor_pdf and top_of_next_higher_floor_pdf >
top_of_bottom_floor_pdf:
            print(f"C2b: For diagrams '{', '.join(map(str, diagram_choices_3))}' The
top of next higher floor should be present, and the elevation should be greater than
the top of bottom floor (C2a).\nTop of next higher floor:
{top_of_next_higher_floor_pdf}\nTop of bottom floor:
{top_of_bottom_floor_pdf}\nElevation Logic matched.\n")
        else:
            print(f"C2b: For diagrams '{', '.join(map(str, diagram_choices_3))}' The
top of next higher floor should be present, and the elevation should be greater than
the top of bottom floor (C2a).\nTop of next higher floor:
{top_of_next_higher_floor_pdf}\nTop of bottom floor: {top_of_bottom_floor_pdf}\nRed
Flag -> Elevation Logic not matched.\n")

    # top of bottom floor = C2a || top of next higher floor = C2b
    if abs(LAG_pdf-top_of_bottom_floor_pdf) > 20:
        print("E: LAG and C2a difference is greater than 20\nFailed -> Underwritter
review required.\n")
    else:
        print("E: LAG and C2a difference is smaller than 20\nPassed -> No underwritter
review required.\n")

    if abs(LAG_pdf-top_of_next_higher_floor_pdf) > 20:
        print("E: LAG and C2b difference is greater than 20\nFailed -> Underwritter
review required.\n")
    else:
        print("E: LAG and C2b difference is smaller than 20\nPassed -> No underwritter
review required.\n")


# Section E measurements used
print("Rule 8\n----------------------------------------------------")
try:
    Section_E_First_Floor_Height = extract_float_value(search_key(data_app,
'ecSectionEFirstFloorHeight'))
```

```
except (ValueError, TypeError):
    Section_E_First_Floor_Height = 0

section_e_measurements_used = False
z
if Section_E_First_Floor_Height > 0:
    section_e_measurements_used = True
    print("A: Section E measurements are used in the application.\n")
else:
    print("A: Section E measurements are not used in the application.\n")

section_E = search_key(data_pdf, 'Section E')
e1a = extract_float_value(search_key(section_E, 'Top of Bottom Floor'))
e1b = extract_float_value(search_second_key(section_E, 'Top of Bottom Floor'))
e2 = extract_float_value(search_key(section_E, 'Next Higher Floor'))

diagram_choices_6 = ["1", "1a", "3", "6", "7", "8"]
diagram_choices_7 = "1b"
diagram_choices_8 = "5"
diagram_choices_9 = ["2", "2a", "2b", "4", "9"]
diagram_choices_10 = ["6", "7", "8", "9"]

if section_e_measurements_used:
    if top_of_bottom_floor_pdf == Section_E_First_Floor_Height:
        print("EC Top of Bottom Floor matches with Section E First Floor Height in
Application.\n")
    else:
        print("Red Flag -> EC Top of Bottom Floor does not match with Section E First
Floor Height in Application.\n")

    if diagramNumber_pdf.lower() in diagram_choices_6:
        if LAG_pdf <= e1b <= LAG_pdf + 2:
            print(f"E1b: For diagram no. in '{', '.join(map(str, diagram_choices_6))}'
The top of bottom floor elevation should be within 2 feet of the LAG, but not below
the LAG.\nTop of Bottom Floor: {e1b}\nLAG: {LAG_pdf}\nElevation Logic matched.\n")
        else:
            print(f"E1b: For diagram no. in '{', '.join(map(str, diagram_choices_6))}'
The top of bottom floor elevation should be within 2 feet of the LAG, but not below
the LAG.\nTop of Bottom Floor: {e1b}\nLAG: {LAG_pdf}\nRed Flag -> Elevation Logic not
matched.\n")

    elif diagramNumber_pdf.lower() == diagram_choices_7:
        if LAG_pdf <= e1b <= LAG_pdf + 6:
            print(f"E1b: For diagram no. in '{diagram_choices_7}' The elevation should
be within 6 feet of the LAG, but not below the LAG.\nTop of Bottom Floor: {e1b}\nLAG:
{LAG_pdf}\nElevation Logic matched.\n")
        else:
            print(f"E1b: For diagram no. in '{diagram_choices_7}' The elevation should be within 6 feet of the LAG, but not below the LAG.\nTop of Bottom Floor: {e1b}\nLAG: {LAG_pdf}\nRed Flag -> Elevation Logic not matched.\n")

    elif diagramNumber_pdf.lower() in diagram_choices_8:
        if LAG_pdf <= e1b <= (LAG_pdf + 20):
            print(f"E1b: For diagram no. in '{diagram_choices_8}' The elevation should be within 20 feet of the LAG, but not below the LAG.\nTop of Bottom Floor: {e1b}\nLAG: {LAG_pdf}\nElevation Logic matched.\n")
        else:
            print(f"E1b: For diagram no. in '{diagram_choices_8}' The elevation should be within 20 feet of the LAG, but not below the LAG.\nTop of Bottom Floor: {e1b}\nLAG: {LAG_pdf}\nRed Flag -> Elevation Logic not matched.\n")

    elif diagramNumber_pdf.lower() in diagram_choices_9:
        if e1a < LAG_pdf:
            print(f"E1b: For diagram no. in '{diagram_choices_9}' the elevation should be below the LAG.\nTop of Bottom Floor: {top_of_bottom_floor_pdf}\nLAG: {LAG_pdf}\nElevation Logic matched.\n")
        else:
            print(f"E1b: For diagram no. in '{diagram_choices_9}' the elevation should be below the LAG.\nTop of Bottom Floor: {top_of_bottom_floor_pdf}\nLAG: {LAG_pdf}\nRed Flag -> Elevation Logic not matched.\n")

    if diagramNumber_pdf.lower() in diagram_choices_10:
        if e2 > e1a:
            print(f"E2: The elevation should be present for building diagrams 6, 7, 8, and 9.  The elevation should be a higher value when compared to E1a.\nTop of next higher floor: {e2}\nTop of bottom floor: {e1a}\nElevation Logic matched.\n")
        else:
            print(f"E2: The elevation should be present for building diagrams 6, 7, 8, and 9.  The elevation should be a higher value when compared to E1a.\nTop of next higher floor: {e2}\nTop of bottom floor: {e1a}\nRed Flag -> Elevation Logic not matched.\n")

    if e1a > 20 or e1b > 20 or e2 > 20:
        print("E: E1a, E1b, or E2 is greater than 20\nFailed -> Underwritter review required.\n")
    else:
        print("E: E1a, E1b, and E2 are smaller than 20\nPassed -> No underwritter review required.\n")


# Section H
#--------------------------------------------------------------------------------
print("Rule 9 - Section H\n----------------------------------------------------")

section_H_in_pdf = search_key(data_pdf, 'Section H')
```

```
h1a_top_of_bottom_floor = extract_float_value(search_key(section_H_in_pdf, 'Top of
Bottom Floor'))
h1b_top_of_next_higher_floor = extract_float_value(search_key(section_H_in_pdf, 'Top
of Next Higher Floor'))

diagram_choices_11 = ['1', '1a', '3', '6', '7','8']
diagram_choices_12 = ['2', '2a', '2b', '4', '9']
diagram_choices_13 = ['2', '2a', '2b', '4', '6', '7', '8', '9']

if diagramNumber_pdf.lower() in diagram_choices_11:
    if LAG_pdf <= h1a_top_of_bottom_floor <= LAG_pdf + 2:
        print(f"H1a: For diagram no. in '{', '.join(map(str, diagram_choices_11))}'
The top of bottom floor elevation should be within 2 feet of the LAG, but not below
the LAG.\nTop of Bottom Floor: {h1a_top_of_bottom_floor}\nLAG: {LAG_pdf}\nElevation
Logic matched.\n")
    else:
        print(f"H1a: For diagram no. in '{', '.join(map(str, diagram_choices_11))}'
The top of bottom floor elevation should be within 2 feet of the LAG, but not below
the LAG.\nTop of Bottom Floor: {h1a_top_of_bottom_floor}\nLAG: {LAG_pdf}\nRed Flag ->
Elevation Logic not matched.\n")

elif diagramNumber_pdf.lower() == '1b':
    if LAG_pdf <= h1a_top_of_bottom_floor <= LAG_pdf + 6:
        print(f"H1a: For diagram no. in '1b' The elevation should be within 6 feet of
the LAG, but not below the LAG.\nTop of Bottom Floor: {h1a_top_of_bottom_floor}\nLAG:
{LAG_pdf}\nElevation Logic matched.\n")
    else:
        print(f"H1a: For diagram no. in '1b' The elevation should be within 6 feet of
the LAG, but not below the LAG.\nTop of Bottom Floor: {h1a_top_of_bottom_floor}\nLAG:
{LAG_pdf}\nRed Flag -> Elevation Logic not matched.\n")

elif diagramNumber_pdf.lower() in diagram_choices_12:
    if LAG_pdf > h1a_top_of_bottom_floor:
        print(f"H1a: For diagram no. in '{', '.join(map(str, diagram_choices_12))}'
The elevation should be below the LAG.\nTop of Bottom Floor:
{h1a_top_of_bottom_floor}\nLAG: {LAG_pdf}\nElevation Logic matched.\n")
    else:
        print(f"H1a: For diagram no. in '{', '.join(map(str, diagram_choices_12))}'
The elevation should be below the LAG.\nTop of Bottom Floor:
{h1a_top_of_bottom_floor}\nLAG: {LAG_pdf}\nRed Flag -> Elevation Logic not
matched.\n")

elif diagramNumber_pdf.lower() == '5':
    if LAG_pdf <= h1a_top_of_bottom_floor <= (LAG_pdf + 20):
        print(f"H1a: For diagram no. in '5' The elevation should be within 20 feet of
the LAG, but not below the LAG.\nTop of Bottom Floor: {h1a_top_of_bottom_floor}\nLAG:
{LAG_pdf}\nElevation Logic matched.\n")
    else:
        print(f"H1a: For diagram no. in '5' The elevation should be within 20 feet of the LAG, but not below the LAG.\nTop of Bottom Floor: {h1a_top_of_bottom_floor}\nLAG: {LAG_pdf}\nRed Flag -> Elevation Logic not matched.\n")

if diagramNumber_pdf.lower() in diagram_choices_13:
    # the next higher floor should be present, and it should be greater than top of bottom floor
    if h1b_top_of_next_higher_floor and h1b_top_of_next_higher_floor > h1a_top_of_bottom_floor:
        print(f"H1b: For diagrams '{', '.join(map(str, diagram_choices_13))}' The top of next higher floor should be present, and the elevation should be greater than the top of bottom floor (H1a).\nTop of next higher floor present.\nTop of next higher floor: {h1b_top_of_next_higher_floor}\nTop of bottom floor: {h1a_top_of_bottom_floor}\nElevation Logic matched.\n")
    else:
        print(f"H1b: For diagrams '{', '.join(map(str, diagram_choices_13))}' The top of next higher floor should be present, and the elevation should be greater than the top of bottom floor (H1a).\nEither Top of next higher floor not present.\nOR\nTop of next higher floor: {h1b_top_of_next_higher_floor}\nTop of bottom floor: {h1a_top_of_bottom_floor}\nRed Flag -> Elevation Logic not matched.\n")


if h1a_top_of_bottom_floor > 20 or h1b_top_of_next_higher_floor > 20:
    print("E: H1a or H1b is greater than 20. Extremely rare case.\nFailed -> Underwritter review required.\n")
else:
    print("E: H1a and H1b are smaller than 20.\nPassed -> No underwritter review required.\n")


#----------------------------------------------------------------
print("\nRule 10\n----------------------------------------------------")

machinery = search_key(data_app, 'machineryOrEquipmentAbove')
c2e_elevation_of_mahinery = extract_float_value(search_key(data_pdf, 'Lowest elevation of Machinery and Equipment (M&E) servicing the building (describe type of M&E and location in section D comments area)'))
e4_top_of_platform = extract_float_value(search_key(data_pdf, 'Top of platform of machinery and/or equipment servicing the building is'))
h2 = search_key(data_pdf, "is all machinery and equipment servicing (as listed in item H2 instructions) elevated to or above the floor indicated by the H2 arrow (shown in the foundation type diagrams at end of section H instructions) for the appropriate building diagram?")
e2 = extract_float_value(search_key(data_pdf, "for building diagrams 6-9 with permanent flood openings provided in section A items B and/or  9 (see pages 1-2 of instructions), the next higher floor (c2.b in applicable building diagram) of the building is"))
```

```
diagram_choices_14 = ['1', '1a', '1b', '3']
diagram_choices_15 = ['2', '2a', '2b', '4', '6', '7', '8', '9']

if machinery:
    print("Machinery or Equipment Above is present in the application.\n")

    if diagramNumber_pdf.lower() in diagram_choices_14:
        if top_of_next_higher_floor_pdf:
            if c2e_elevation_of_mahinery >= top_of_next_higher_floor_pdf:
                print(f"For diagrams '{', '.join(map(str, diagram_choices_14))}' The
elevation of machinery and equipment should be equal or greater than the top of next
higher floor.\nElevation of machinery and equipment: {c2e_elevation_of_mahinery}\nTop
of next higher floor: {top_of_next_higher_floor_pdf}\nElevation Logic matched.\n")
            else:
                print(f"For diagrams '{', '.join(map(str, diagram_choices_14))}' The
elevation of machinery and equipment should be equal or greater than the top of next
higher floor.\nElevation of machinery and equipment: {c2e_elevation_of_mahinery}\nTop
of next higher floor: {top_of_next_higher_floor_pdf}\nRed Flag -> Elevation Logic not
matched.\n")

        elif not top_of_next_higher_floor_pdf:
            if c2e_elevation_of_mahinery >= top_of_bottom_floor_pdf + 8:
                print(f"Top of next higher floor is not present in EC. For diagrams
'{', '.join(map(str, diagram_choices_14))}' The elevation of machinery and equipment
should be at least 8 feet higher than the top of bottom floor.\nElevation of machinery
and equipment: {c2e_elevation_of_mahinery}\nTop of bottom floor:
{top_of_bottom_floor_pdf}\nElevation Logic matched.\n")
            else:
                print(f"Top of next higher floor is not present in EC. For diagrams
'{', '.join(map(str, diagram_choices_14))}' The elevation of machinery and equipment
should be at least 8 feet higher than the top of bottom floor.\nElevation of machinery
and equipment: {c2e_elevation_of_mahinery}\nTop of bottom floor:
{top_of_bottom_floor_pdf}\nRed Flag -> Elevation Logic not matched.\n")

        if e4_top_of_platform >= e1b + 8:
            print(f"For diagrams '{', '.join(map(str, diagram_choices_14))}' The top
of platform of machinery and/or equipment servicing the building should be at least 8
feet higher than the top of bottom floor.\nTop of platform: {e4_top_of_platform}\nTop
of bottom floor: {e1b}\nElevation Logic matched.\n")
        else:
            print(f"For diagrams '{', '.join(map(str, diagram_choices_14))}' The top
of platform of machinery and/or equipment servicing the building should be at least 8
feet higher than the top of bottom floor.\nTop of platform: {e4_top_of_platform}\nTop
of bottom floor: {e1b}\nRed Flag -> Elevation Logic not matched.\n")

        if h2.lower() == "yes":
            print("H2 is marked as 'Yes' in the EC\n")
        elif h2.lower() == "no":
            print("H2 is marked as 'No' in the EC\n")

        else:
            print("H2 is not marked in the EC\n")


    elif diagramNumber_pdf.lower() in diagram_choices_15:

        if c2e_elevation_of_mahinery >= top_of_next_higher_floor_pdf:
            print(f"For diagrams '{', '.join(map(str, diagram_choices_15))}' The
elevation of machinery and equipment should be equal or greater than the top of next
higher floor.\nElevation of machinery and equipment: {c2e_elevation_of_mahinery}\nTop
of next higher floor: {top_of_next_higher_floor_pdf}\nElevation Logic matched.\n")
        else:
            print(f"For diagrams '{', '.join(map(str, diagram_choices_15))}' The
elevation of machinery and equipment should be equal or greater than the top of next
higher floor.\nElevation of machinery and equipment: {c2e_elevation_of_mahinery}\nTop
of next higher floor: {top_of_next_higher_floor_pdf}\nRed Flag -> Elevation Logic not
matched.\n")

        if e4_top_of_platform >= e2:
            print(f"For diagrams '{', '.join(map(str, diagram_choices_15))}' The top
of platform of machinery and/or equipment servicing the building should be at least 8
feet higher than the top of bottom floor.\nTop of platform: {e4_top_of_platform}\nTop
of bottom floor: {e2}\nElevation Logic matched.\n")
        else:
            print(f"For diagrams '{', '.join(map(str, diagram_choices_15))}' The top
of platform of machinery and/or equipment servicing the building should be at least 8
feet higher than the top of bottom floor.\nTop of platform: {e4_top_of_platform}\nTop
of bottom floor: {e2}\nRed Flag -> Elevation Logic not matched.\n")

        if h2.lower() == "yes":
            print("H2 is marked as 'Yes' in the EC\n")

        elif h2.lower() == "no":
            print("H2 is marked as 'No' in the EC\n")

        else:
            print("H2 is not marked in the EC\n")

    elif diagramNumber_pdf.lower() == '5':
        if c2e_elevation_of_mahinery >= top_of_bottom_floor_pdf:
            print(f"For diagrams '5' The elevation of machinery and equipment should
be equal or greater than the top of bottom floor.\nElevation of machinery and
equipment: {c2e_elevation_of_mahinery}\nTop of bottom floor:
{top_of_bottom_floor_pdf}\nElevation Logic matched.\n")
        else:
            print(f"For diagrams '5' The elevation of machinery and equipment should be equal or greater than the top of bottom floor.\nElevation of machinery and equipment: {c2e_elevation_of_mahinery}\nTop of bottom floor: {top_of_bottom_floor_pdf}\nRed Flag -> Elevation Logic not matched.\n")

        if e4_top_of_platform >= e1b:
            print(f"For diagrams '5' The top of platform of machinery and/or equipment servicing the building should be higher than the top of bottom floor.\nTop of platform: {e4_top_of_platform}\nTop of bottom floor: {e1b}\nElevation Logic matched.\n")
        else:
            print(f"For diagrams '5' The top of platform of machinery and/or equipment servicing the building should be higher than the top of bottom floor.\nTop of platform: {e4_top_of_platform}\nTop of bottom floor: {e1b}\nRed Flag -> Elevation Logic not matched.\n")

        if h2.lower() == "yes":
            print("H2 is marked as 'Yes' in the EC\n")

        elif h2.lower() == "no":
            print("H2 is marked as 'No' in the EC\n")

        else:
            print("H2 is not marked in the EC\n")

else:
    print("Machinery or Equipment Above is not present in the application.\n")
```