

# L'Elaboratore del Collaudo

PROGETTO FORMATIVO

STANCIU NICOLAS

## Sommario

-Obiettivi:.....	2
-Come realizzarlo .....	2
PARTE 1: Realizzata in C++ e Python.....	2
PARTE 2: Realizzata in Python.....	2
-Realizzazione pratica del lavoro .....	3
PARTE 1: Realizzata in C++.....	3
-Librerie usate .....	3
-Correzione nome del file .....	3
-Separazione campi di ogni riga del file .....	3
-Suddivisione valori ed identificativi dei dati .....	4
-Preparazione dei dati alla conversione .....	4
-Conversione dei dati .....	4
-Riscrittura dei dati con le loro unità di misura .....	5
-Spostamento file finale .....	5
-Esempio del file CSV risultante .....	6
PARTE 1: Rifatta in Python.....	6
-Librerie usate .....	6
-Apertura e lettura dal file .....	7
-Conversione dati .....	7
-Riscrittura dei dati con le loro unità di misura .....	7
PARTE 2 (versione precedente): Realizzata in Python .....	8
-Librerie usate .....	8
-Definizione dei profili di stile delle celle .....	8
-Definizione della struttura e dei contenuti delle righe .....	8
-Gestione altezza riga ed unione celle .....	9
-Assegnazione contenuti e applicazione stili alle celle.....	9
-Estrazione dati dal nome del file .....	9
-Estrazione dati dal file .....	9
-Estrazione degli elementi che compongono le versioni .....	10
-Ordinamento degli elementi che compongono le versioni.....	10
-Uniformazione dei nomi dei campi.....	10
-Controllo dei campi per l'inserimento dei dati .....	11
-Controllo su dati booleani .....	11
-Inserimento dati nel template XLSX.....	11
PARTE 2 (versione ottimizzata): Realizzata in Python .....	12
-Salvataggio indirizzi delle celle unite .....	12
-Inserimento dati nel template XLSX.....	12
-Esempio del file XLSX risultante .....	13
-Requisiti Python.....	13

## -Obiettivi:

L'obiettivo del progetto è processare i dati grezzi provenienti dal collaudo, inserirli in un file CSV e successivamente trasferirli in un nuovo template XLSX, opportunamente formattato e impaginato. I dati analizzati vengono sovrascritti nel file CSV di origine, insieme al relativo identificativo. Grazie a questo identificativo, il programma può collocare i dati nelle celle corrette del file XLSX. Il file finale viene poi spostato nella cartella dedicata al cliente. Il programma deve inoltre essere in grado di gestire dati provenienti da più file CSV diversi, inserendoli automaticamente nel template XLSX.

## -Come realizzarlo

Per realizzare il progetto è necessario suddividerlo in due fasi e seguire i passaggi elencati per entrambe

### PARTE 1: Realizzata in C++ e Python

- 1 - Eseguire uno script PS1 per ricavare più file CSV contenenti i dati rilevati da un collaudo
- 2 – Controllare la correttezza dei nomi dei file e se non coerenti modificarli
- 3 – Analizzare ogni file riga per riga estraendo i dati necessari ed inserendoli in una struttura adeguata
- 4 – Convertire i dati fuori scala in modo da renderli più leggibili
- 5 – Scorrere la struttura e riscrivere i dati contenuti, insieme alle relative unità di misura, all'interno di un nuovo file CSV
- 6 – Una volta che i dati sono stati convertiti ed il file CSV è stato formattato, il file viene spostato in una cartella dedicata, così da renderlo disponibile per la seconda fase del progetto

### PARTE 2: Realizzata in Python

- 1 – Definire una struttura per memorizzare i profili di stile dei testi
- 2 – Definire una struttura che contenga i dettagli di ogni riga: altezza, celle da unire, contenuto e profilo di stile
- 3 – Analizzare e modificare la struttura delle righe secondo i parametri indicati. Il template xlsx finale deve essere poi spostato in una cartella dedicata.
- 4 – Memorizzare in una struttura adeguata i dati contenuti nel file csv formattato
- 5 – Ricavare eventuali dati non presenti direttamente nel CSV o separati all'interno di esso
- 6 – Analizzare gli identificativi presenti nel template XLSX ed i campi nella struttura per trovare le corrispondenze ed inserire i dati nelle rispettive celle
- 7 – Salvare il file finale all'interno di una cartella dedicata

## -Realizzazione pratica del lavoro

### PARTE 1: Realizzata in C++

#### -Librerie usate

Per la realizzazione del programma in C++ sono state utilizzate le seguenti librerie:

- “iostream”: per la gestione dell'input/output da console
- “filesystem”: per la gestione dei file e delle cartelle
- “fstream”: per la lettura e la scrittura dei file CSV.
- “vector”: per la gestione di array dinamici, utilizzati per memorizzare dati e tipi di dati letti dai file.
- “map”: per associare i nomi dei campi alle rispettive unità di misura.

#### -Correzione nome del file

```
for (int index{0}; index < filename.length(); index++) {  
    if (filename[index] == '.') {  
        num_dots++;  
    }  
  
    if (!std::isalnum(filename[index]) && filename[index] != '_') {  
        if (filename[index] == '.' && num_dots <= 1) continue;  
        filename[index] = '_';  
    }  
}
```

Tramite un ciclo FOR, ogni carattere del nome del file viene analizzato. Se il ciclo rileva caratteri diversi da lettere e numeri, questi vengono sostituiti con un carattere underscore (“\_”). Inoltre, se nel nome del file è presente più di un punto, anche i punti successivi al primo vengono sostituiti con underscore.

#### -Separazione campi di ogni riga del file

```
for (int index{0}; index <= line.length(); index++) {  
    if (index == line.length() || line[index] == separator) {  
        line_parts.push_back(line.substr(start_index, index - start_index));  
        start_index = index + 1;  
    }  
}
```

push\_back = metodo della libreria Vector utilizzato per aggiungere un elemento alla fine di un vettore. Accetta come parametro un elemento dello stesso tipo del vettore su cui viene chiamato.

substr = metodo della libreria String che estrae una parte della stringa, dato l'indice di inizio e la lunghezza del segmento da estrarre.

Per estrarre le parole da ogni campo di una riga, un ciclo FOR analizza i caratteri e, al rilevamento del separatore o della fine riga, estrae il segmento tra i separatori o tra l'ultimo separatore e la fine.

### -Suddivisione valori ed identificativi dei dati

```
if (line_parts.at(0) == "TIME") {
    for (std::string str : line_parts) {
        data_type.push_back(str);
    }
} else {
    for (std::string str : line_parts) {
        data.push_back(str);
    }
}
```

Viene verificata la prima parola dell'array che contiene i campi della riga. Se questa parola è uguale a "TIME", significa che la riga rappresenta gli identificativi dei campi e non i dati grezzi. In questo modo, è possibile memorizzare separatamente gli identificativi in un array e i dati grezzi in un altro.

### -Preparazione dei dati alla conversione

```
for (int data_field{0}; data_field < data.size(); data_field++) {
    std::string new_data{ data.at(data_field) };
    while (new_data.find('.') != -1) {
        new_data.erase(new_data.find('.'), 1);
    }
    data.at(data_field) = new_data;
}
```

find = metodo della libreria String che cerca una sottostringa o un carattere all'interno della stringa e restituisce l'indice della prima occorrenza, oppure -1 se non viene trovato.

erase = metodo della libreria String che rimuove una parte della stringa, specificando l'indice di inizio e la lunghezza del segmento da eliminare.

Tutti i dati memorizzati vengono analizzati tramite un ciclo FOR e, per ciascun dato, vengono rimossi i punti. Questo passaggio facilita le successive operazioni di conversione sui dati.

### -Conversione dei dati

```
for (int data_field{0}; data_field < data.size(); data_field++) {
    for (auto field : fields_to_modify){
        if (data_type.at(data_field).find(field) != -1){
            double value = std::stod(data.at(data_field)) * 0.000001;
            std::string new_data = std::to_string(value);
            if (new_data.find('.') != -1){
                new_data = new_data.substr(0, new_data.find('.') + 2);
            }
            data.at(data_field) = new_data;
        }
    }
}
```

I dati raccolti vengono esaminati scorrendo la struttura in cui sono memorizzati. I valori che necessitano di conversione vengono trasformati in double, successivamente moltiplicati per 0,000001 e infine riconvertiti in stringa.

### -Riscrittura dei dati con le loro unità di misura

```
for (int data_field{0}; data_field < data_type.size(); data_field++) {
    line.clear();
    line += data_type.at(data_field) + ";";
    line += data.at(data_field) + ";";
    for (auto mu : measurement_units){
        if (data_type.at(data_field).find(mu.first) != -1) line += mu.second + ";";
    }
    if (data_field == data_type.size() - 1) {
        output_file << line;
    } else {
        output_file << line << std::endl;
    }
}
```

clear = metodo della libreria String che elimina tutti i caratteri dalla stringa, rendendola vuota.

Dopo aver ottenuto e convertito tutti i dati, il programma procede a riscriverli nel file CSV originale. Scorrendo una mappa che associa a ciascun tipo di dato la relativa unità di misura, il programma collega ogni dato estratto dalla struttura ai rispettivi identificativi e, se previsto, assegna loro l'unità di misura. In questo modo vengono create stringhe che includono l'identificativo del campo, il valore e l'unità di misura, che vengono poi scritte all'interno del file CSV.

### -Spostamento file finale

```
bool moveFile(std::string filename, std::string destination){
    try {
        std::filesystem::create_directory(destination);

        std::filesystem::rename(filename, destination + "/" + filename);
    } catch (const std::filesystem::filesystem_error& e) {
        return false;
    }
    return true;
}
```

create\_directory = metodo della libreria Filesystem che crea una cartella nel percorso specificato dal path fornito.

rename = metodo della libreria Filesystem che consente di rinominare o spostare un file specificando il nuovo nome o percorso.

Si avvia una funzione che riceve come parametri il nome del file e la cartella in cui lo si vuole inserire. Viene utilizzato un try-catch per poter gestire l'eccezione lanciata nel caso in cui non si riesca ad effettuare lo spostamento.

## -Esempio del file CSV risultante

TIME	2025-05-12T16:06:48+02:00	
EVENT		
TempMandata_Mevo	21.9	'C
TempRitorno_Mevo	23.3	'C
VersioneSoftware1_Mevo	3	
VersioneSoftware2_Mevo	1	
VersioneSoftware3_Mevo	1	
TempAcs	34.5	'C
Alta_Pressione_Mevo	12.1	bar
Bassa_Pressione_Mevo	6.0	bar
PosValvola_Mevo	394	stps
VelCompressore_Mevo	37.7	rps
Portata_ACS	15.0	l/h
Portata_Impianto	2566.0	l/h
VersioneOS_1_Mevo	6	
VersioneOS_2_Mevo	0	
VersioneOS_3_Mevo	4	

## PARTE 1: Rifatta in Python

La parte 1, precedentemente realizzata in C++, è stata ora riscritta in Python. Le funzionalità implementate e il risultato finale rimangono invariati rispetto alla versione precedente.

### -Librerie usate

Per la realizzazione del programma in Python sono state usate le seguenti librerie:

- “pathlib”: utilizzata per la gestione dei percorsi di file e cartelle
- “csv”: utilizzata per la lettura e la scrittura dei file CSV
- “os”: utilizzata per operazioni sul filesystem, come la creazione di cartelle

## -Apertura e lettura dal file

```
with open(current_file.name, newline = '', encoding = 'utf-8') as csvfile:
    reader = csv.reader(csvfile, delimiter = ',')
    for row in reader:
        if len(row) > 0:
            if row[0].lower() == 'time':
                fields_name = row
            elif len(fields_data) == 0:
                fields_data = row
```

reader = funzione della libreria CSV che legge un file CSV e restituisce ogni riga come una lista di valori, separando i campi in base al delimitatore specificato

Dopo aver aperto il file CSV, il programma suddivide ogni riga utilizzando il delimitatore. Se la riga contiene almeno un elemento, verifica se si tratta dell'intestazione dei campi o dei dati. Una volta distinti i due tipi di riga, i relativi valori vengono memorizzati in una struttura dati.

## -Conversione dati

```
for key, value in csv_data.items():
    for field_to_modify in fields_to_modify:
        if field_to_modify in key.lower():
            csv_data[key] = str(float(value))
            fields_modified = True
            break
```

Alcuni dati vengono convertiti in float per essere riportati su una scala più leggibile e successivamente riconvertiti in stringa. Questa operazione viene eseguita identificando i campi da convertire tramite un array che contiene i nomi dei parametri interessati.

## -Riscrittura dei dati con le loro unità di misura

```
for field_name, field_data in csv_data.items():
    measurement_unit = ''
    for type_unit, unit in measurement_units.items():
        if type_unit in field_name.lower():
            measurement_unit = unit
            field_name = field_name.replace('_Mevo', '')
            writer.writerow([field_name, field_data, measurement_unit])
```

writerow = metodo della libreria CSV che scrive una riga nel file CSV, accettando una lista di valori da inserire come campi nella riga.

Scorrendo un dizionario che associa a ciascun tipo di dato la relativa unità di misura, il programma collega ogni dato estratto dalla struttura ai rispettivi campi e assegna l'unità di misura appropriata, se prevista. Ogni riga del file finale sarà quindi composta dal nome del campo, dal valore del dato e dalla sua unità di misura.



## PARTE 2 (versione precedente): Realizzata in Python

### -Librerie usate

Per la realizzazione del programma in Python sono state usate le seguenti librerie:

- "openpyxl":
- "PIL":
- "pathlib":
- "-re":

### -Definizione dei profili di stile delle celle

```
cells_profiles = {  
    0: {  
        'alignment': Alignment(wrap_text=True, horizontal='left', vertical='center'),  
        'font': Font(size = 7.5),  
        'border': Border(left = thin, top = thin, right = thin, bottom = thin),  
        'fill': PatternFill()  
    },  
}
```

Viene creato un dizionario in cui ogni chiave rappresenta l'ID di un profilo, mentre il valore associato è un altro dizionario. Quest'ultimo contiene, come chiavi, i nomi dei parametri da modificare e, come valori, le relative impostazioni o modifiche da applicare.

### -Definizione della struttura e dei contenuti delle righe

```
cells_layout = []  
cells_layout.append({'ROW_HEIGHT': [15], 'B-N': ['DATI GENERALI', 5]})  
cells_layout.append({'ROW_HEIGHT': [25], 'B-G': ['', 12], 'H-N': ['', 12]})
```

Viene creata una lista in cui ogni elemento è un dizionario. In ciascun dizionario, la chiave rappresenta un identificativo che indica il tipo di operazione da eseguire: ad esempio, 'ROW\_HEIGHT' specifica che deve essere modificata l'altezza della riga, mentre una chiave nel formato 'X-Y' indica le colonne da unire. Il valore associato alla chiave contiene i dettagli dell'operazione: per 'ROW\_HEIGHT' è indicata la nuova altezza della riga, mentre per 'X-Y' sono specificati il contenuto della cella e il profilo di stile da applicare.

## -Gestione altezza riga ed unione celle

```
if key == 'ROW_HEIGHT':  
    ws.row_dimensions[cell_row].height = value[0]  
else:  
    start_merge_cell_col = key.split('-')[0] if '-' in key else key  
    end_merge_cell_col = key.split('-')[1] if '-' in key else key  
    ws.merge_cells(f'{start_merge_cell_col}{cell_row}:{end_merge_cell_col}{cell_row}')
```

ws = oggetto Workbook che rappresenta il foglio XLSX di lavoro

merge\_cells = metodo della libreria Openpyxl che unisce più celle, accettando come parametro l'intervallo (riga e colonna iniziale e finale) da unire.

Ogni chiave del dizionario viene analizzata e, in base al suo valore, si procede a modificare l'altezza della riga oppure a unire più celle. Per effettuare l'unione, dalla chiave nel formato 'X-Y' vengono estratte le colonne iniziale e finale, che identificano l'intervallo delle celle da unire.

## -Assegnazione contenuti e applicazione stili alle celle

```
ws[cell_address] = value[0]  
ws[cell_address].alignment = cells_profiles[value[1]]['alignment']  
ws[cell_address].font = cells_profiles[value[1]]['font']  
ws[cell_address].fill = cells_profiles[value[1]]['fill']
```

Il contenuto viene assegnato alle celle e, in base al profilo associato, vengono applicati gli stili di formattazione corrispondenti.

## -Estrazione dati dal nome del file

```
csv_data = {}  
info_parts = current_filename.split('_')  
csv_data['Matricola'] = info_parts[1]  
csv_data['Commissa'] = info_parts[0]  
csv_data['Conferma'] = info_parts[1][:5]
```

Le informazioni contenute nel nome del file CSV formattato vengono estratte e aggiunte, con il rispettivo identificativo, al dizionario che raccoglie tutti i dati.

## -Estrazione dati dal file

```
for line in current_csv:  
    line_parts = line.split(';')  
    if line_parts[0].lower() == 'time':  
        line_parts[0] = 'Data'  
        line_parts[1] = line_parts[1][:line_parts[1].find('T')]  
    csv_data[line_parts[0].replace('\\"', '')] = line_parts[1] + ' ' + line_parts[2]
```

I dati con le unità di misura contenuti nel file CSV formattato vengono estratti ed aggiunti, con il loro rispettivo identificativo, al dizionario che raccoglie tutti i dati. Dal campo "time" viene estratta la data.

### -Estrazione degli elementi che compongono le versioni

```
versions = {}
for key, value in csv_data.items():
    if key.find('Versione') != -1:
        version_order = ''.join(ch if ch.isdigit() else ' ' for ch in key)
        versions.setdefault(key.replace(version_order, ''), {})[version_order]=value
keys_to_delete = [key for key in csv_data if 'Versione' in key]
for key in keys_to_delete : csv_data.pop(key, None)
```

Scorrendo il dizionario che contiene tutti i dati, le chiavi che includono la parola "Versione" vengono individuate. Da ciascuna chiave viene estratto il numero che ne indica l'ordine, e la coppia (numero, valore) viene inserita come valore in un dizionario dedicato alle versioni, che ha come chiave il nome della versione senza il numero. Successivamente, tutte le chiavi contenenti "Versione" vengono rimosse dal dizionario principale dei dati

### -Ordinamento degli elementi che compongono le versioni

```
for key, value in versions.items():
    version_size = max(int(size) for size in value)
    versions_correct_order = [None] * version_size
    version_num = ''
    for version_order, version_value in value.items():
        versions_correct_order[int(version_order) - 1] = version_value
    for num_version in versions_correct_order:
        if num_version : version_num += num_version + '.'
    version_num = version_num[:-1]
    csv_data[key] = version_num
```

Per ogni gruppo di versioni, i valori vengono ordinati in base al numero estratto e concatenati in un'unica stringa separata da punti, che viene poi reinserita nel dizionario principale con la chiave corrispondente al tipo di versione.

### -Uniformazione dei nomi dei campi

```
key_name = key
key_name = ''.join(ch if ch.isalnum() else ' ' for ch in key_name)
for index in range(0, len(key_name)):
    if key_name[index].isupper() and index != 0 and key_name[index - 1].islower():
        key_name = key_name[:index] + ' ' + key_name[index:]
key_name = ''.join(ch.lower() if ch.isalpha() else ch for ch in key_name)
field_to_search = key_name.split(' ')
```

Ogni chiave del dizionario dei dati viene normalizzata: tutti i caratteri non alfanumerici sono sostituiti da uno spazio, le parole in formato camel case vengono separate, tutte le lettere sono convertite in minuscolo e le parole risultanti vengono inserite in un array.

## -Controllo dei campi per l'inserimento dei dati

```
if is_first_in_merged_cell(ws, current_cell_address):
    for field_part in field_to_search:
        field_name = ws[current_cell_address].value.lower() if ws[current_cell_address].value else ''
        if not(ws[current_cell_address].value and re.search(rf'\b{field_part}', field_name) != None):
            can_put_data = False
            break
    else:
        continue
```

Se la cella analizzata corrisponde alla prima cella di un intervallo di celle unite, si verifica che ogni parola del campo in esame sia presente nel campo corrispondente del template; solo se tutte le parole sono contenute, si autorizza l'inserimento del dato.

## -Controllo su dati booleani

```
if ws[first_merged_col_address].value.lower().find('verifica') != -1:
    value = value.replace('\n', '').replace('\r', '').strip()
    if value == '0' or value == 0:
        value = 'No'
    elif value == '1' or value == 1:
        value = 'Si'
```

Se la cella che identifica il campo da inserire contiene la parola “verifica” nel nome, il dato da inserire viene interpretato come booleano. In questo caso, il valore viene convertito in “No” se è 0 (False) oppure in “Si” se è 1 (True).

## -Inserimento dati nel template XLSX

```
if can_put_data is True:
    for next_col in range(col + 1, ord(template_end_col) - 65):
        cell_address_to_put_data = f'{chr(next_col + 66)}{row}'
        if is_first_in_merged_cell(ws, cell_address_to_put_data):
            if ws[cell_address_to_put_data].value is None:
                ws[cell_address_to_put_data] = value
                break
            elif re.search(r':\s*$', ws[cell_address_to_put_data].value) :
                ws[cell_address_to_put_data] = ws[cell_address_to_put_data].value + ' ' + value
                break
```

Scorrendo le colonne successive a quella del campo trovato, se la cella è la prima di un gruppo di celle unite ed è vuota oppure contiene una parola che termina con i due punti, il dato viene inserito in quella cella.

## PARTE 2 (versione ottimizzata): Realizzata in Python

Per ridurre i tempi di esecuzione del programma, ho ottimizzato la procedura di scrittura nel template XLSX. Ora il template viene scansionato non più cella per cella, ma direttamente da una cella unita all'altra. Inoltre, sono stati aggiunti controlli per escludere automaticamente le celle non rilevanti.

### -Salvataggio indirizzi delle celle unite

```
for merged_range in ws.merged_cells.ranges:
    merged_range_parts = str(merged_range).split(':')
    if len(merged_range_parts) < 2: merged_range_parts.append(merged_range_parts[0])
    merged_cells_ranges[merged_range_parts[0]] = merged_range_parts[1]
```

Durante la scansione delle celle unite nel foglio Excel, per ogni intervallo unito viene salvato in un dizionario l'indirizzo della prima cella come chiave e quello dell'ultima cella dell'intervallo come valore. Questo permette di identificare rapidamente i limiti di ciascuna area unita nel foglio di lavoro.

### -Inserimento dati nel template XLSX

```
next_merged_cell_address = chr(ord(last_merged_col_address[0]) + 1) + last_merged_col_address[1:]
while ord(next_merged_cell_address[0]) < ord(template_end_col):
    if not(ws[next_merged_cell_address].value):
        ws[next_merged_cell_address] = value
        break
    elif re.search(r':\s*$', ws[next_merged_cell_address].value):
        ws[next_merged_cell_address] = ws[next_merged_cell_address].value + ' ' + value
        break
    next_merged_cell_col = chr(ord(merged_cells_ranges[next_merged_cell_address][0]) + 1)
    next_merged_cell_row = str(merged_cells_ranges[next_merged_cell_address][1:])
    next_merged_cell_address = next_merged_cell_col + next_merged_cell_row
```


Dopo aver individuato la fine di una cella unita, il codice calcola l'indirizzo della cella immediatamente successiva sulla stessa riga. Entra quindi in un ciclo che scorre verso destra, controllando ogni cella fino al limite della tabella (definito da `template_end_col`).

Se trova una cella vuota, vi inserisce il valore desiderato e interrompe il ciclo.

Se invece la cella contiene già un valore che termina con i due punti (:), aggiunge il nuovo valore in coda, separandolo con uno spazio, e poi esce dal ciclo.

Se nessuna delle condizioni precedenti è soddisfatta, aggiorna l'indirizzo della cella successiva da controllare, spostandosi ancora a destra, utilizzando le informazioni sulle celle unite memorizzate nella mappa. In questo modo, il codice garantisce che i dati vengano inseriti nella prima cella disponibile dopo un intervallo di celle unite, oppure concatenati correttamente se necessario.

## -Esempio del file XLSX risultante

REPORT AUTOMATICO COLLAUDO CHILLER MEVO				Mod. PROD. 03 Rev. 0 del 05/06/2025
		G.S.I. srl Via dell'Artigianato 44 31047 Ponte di Piave (TV) Tel 0422 289828 .Fax. 0422 759905		
Matricola	xxxxxxxxxxx	Data	2025-06-16	
Commessa	xxxxx	Conferma	xxxxx	
DATI GENERALI				
Versione software	3.1.3			
Versione OS	6.0.4			
POMPE A BORDO	108 - 112	116 - 120		
Pompa IMPIANTO	PARA R 15-130 / 9-87 / IPWM2		PARA MAXO R 25-180-10-F22 GSY	
Pompa ACS	PARA R 15-130 / 9-87 / IPWM2		PARA MAXO R 25-180-10-F22 GSY	
Portata IMPIANTO	3617.9996 l/h			
Portata ACS	3309.0 l/h			
SONDE E TRASDUTTORI	LETTURA			
Temperatura Mandata Impianto	18.4 °C			
Temperatura Ritorno Impianto	19.6 °C			
Temperatura ACS	37.0 °C			
Alta Pressione	12.7 bar			
Bassa Pressione	5.9 bar			
INVERTER COMPRESSORE				
Velocità Compressore	37.2 rps			
Verifiche	Verificato			
Verifica EEV 1 Movimento	Si			
Verifica EEV 2 Movimento	Si			
Verifica Resistenza Vaschetta Funzionamento	Si			
Verifica Bobina 4 VIE Funzionamento	No			
Verifica Sonde NTC macchina	Si			
Verifica TRASDUTTORI	Si			
Gas Refrigerante	R290	Qta [kg]: 0		
VENTILATORI				
Verifica Ventilatore 1 (alto)	Modello:VHE01195338	Comando 0..10Vdc: Si		
Verifica Ventilatore 2 (basso)	Modello:VHE01195338	Comando 0..10Vdc: Si		

## -Requisiti Python

Versione = 3.13.4

Libreria 'openpyxl' = installazione tramite cmd con comando "pip install openpyxl"

Libreria 'pillow' = installazione tramite cmd con comando "pip install pillow"