greenhouse

Version 3.0.0

Table of Contents

Inhalt:

Gewächshaus-Steuerungssystem

Dies ist die technische Dokumentation für das **Gewächshaus-Steuerungssystem** . Dieses Projekt bietet ein komplettes Überwachungs- und Steuerungssystem für ein Gewächshaus mit Fokus auf den Anbau von Hanfblüten für die Pharma- und Teeindustrie.

Projektübersicht

Das Projekt wurde in mehreren Abschnitten entwickelt:

- **Abschnitt 1-2:** Analyse der Steuerung
- **Abschnitt 3:** Temperatur- und Luftfeuchtemessung
- **Abschnitt 4:** Erweiterung der Anzeige
- **Abschnitt 5:** Integration des Helligkeitssensors
- **Abschnitt 6:** Helligkeitssteuerung
- **Abschnitt 7:** Datenspeicherung

Analyse der Steuerung

In den ersten beiden Wochen des Projekts wurde eine umfassende Analyse der Steuerungshardware und -software durchgeführt. Dabei entstand ein detailliertes Blockschaltbild, das die gesamte geplante Steuerung aller Projektabschnitte darstellt.

Zudem wurde eine Übersicht über die Messbereiche und Toleranzen aller verwendeten Sensoren erstellt und entsprechende Datenblätter hinzugefügt.

Temperatur- und Luftfeuchtemessung

Im dritten Projektabschnitt wurde die Temperaturmessung mit dem Sensor DHT11 und der 7-Segment-LED-Anzeige in Betrieb genommen und im Dauerbetrieb getestet.

Die Temperaturwerte werden mit ausreichender Genauigkeit angezeigt. Ein entsprechendes Protokoll dokumentiert diesen Prozess.

Im vierten Projektabschnitt wurde der Luftfeuchtesensor des DHT11 integriert. Die Daten dieses Sensors werden mit erfasst und in sinnvoller Genauigkeit angezeigt.

Erweiterung der Anzeige

Die erfassten Werte für Temperatur und Luftfeuchte werden zusätzlich zur Siebensegmentanzeige gemeinsam in einem LCD (2 x 16) angezeigt.

Das LCD-Display bietet eine bessere Lesbarkeit und ermöglicht die gleichzeitige Darstellung beider Messwerte mit Beschriftung.

Integration des Helligkeitssensors

Da die Gärtnerei Hanfblüten zur Teeherstellung und für die Pharmaindustrie anbaut, muss die Beleuchtung für blühenden Hanf sichergestellt werden. Dazu wurde ein Helligkeitssensor in die Steuerung integriert.

Die Helligkeit wird bewertet und die Bewertung mittels verschiedener Symbole auf der Matrixanzeige signalisiert:

- ☀ **Sonnensymbol** : Zeigt ausreichende Helligkeit an (>100 Lux)
- ☽ **Mondsymbol** : Zeigt niedrige Helligkeit an (<100 Lux)

Helligkeitssteuerung

Auf Grundlage der bewerteten Helligkeit wurde eine Lichtsteuerung mit Leistungskreis (Relais) implementiert. Dabei wird die aktuelle Tageszeit (Sonnenauf- und -untergang) berücksichtigt.

Die Systemzeit wird kontinuierlich über einen lokalen Zeitserver (10.254.5.115) aktualisiert, um eine präzise zeitgesteuerte Beleuchtung zu gewährleisten.

Datenspeicherung

Alle gemessenen Werte werden mit Zeitstempel in einer SQLite-Datenbank gespeichert. Dies umfasst:

- Zeitstempel (synchronisiert über NTP)
- Temperaturwerte
- Luftfeuchtigkeitswerte
- Helligkeitswerte

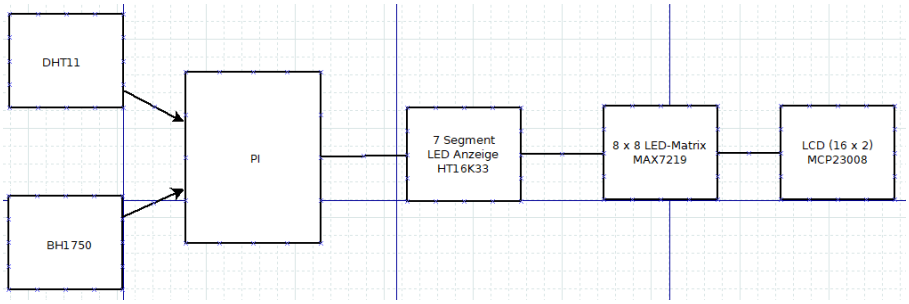
Diese Datenbank ermöglicht eine spätere Auswertung der Umgebungsbedingungen und dient als Grundlage für die Optimierung der Wachstumsbedingungen.

Problemstellung

Das Gewächshaus-Steuerungssystem wurde entwickelt, um folgende Herausforderungen zu lösen:

- Kontinuierliche Überwachung der Umgebungsparameter für optimales Pflanzenwachstum
- Automatische Steuerung der Beleuchtungsbedingungen für Hanfblüten
- Präzise Datenerfassung und -speicherung für Analyse und Optimierung
- Benutzerfreundliche Visualisierung der Systemzustände für das Gärtnereipersonal
- Dauerhafte Dokumentation der Umgebungsbedingungen für Qualitätssicherung

Systemaufbau



Systemdiagramm des Gewächshaus-Steuerungssystems

Hauptfunktionen

Das System umfasst folgende Hauptfunktionen:

- Erfassung von Temperatur- und Luftfeuchtigkeitsdaten über den DHT11-Sensor
- Messung des Umgebungslichts mit einem Helligkeitssensor
- Speicherung aller Messwerte mit Zeitstempel in einer SQLite-Datenbank
- Anzeige der aktuellen Temperatur und Luftfeuchtigkeit auf einem LCD-Display
- Visualisierung von Tag/Nacht-Zuständen über ein 8x8 LED-Matrix-Display
- Anzeige der numerischen Messwerte auf einem 7-Segment-Display
- Synchronisation der Systemzeit über einen lokalen NTP-Server
- Umfassende Protokollierung aller Aktivitäten und Fehler

Komponenten

Das Projekt besteht aus folgenden Software-Komponenten:

- Ein benutzerdefiniertes Protokollierungsmodul (*school_logging*) für flexible und farbige Protokollierung
- Ein Messdaten-Modul (*greenhouse/messdaten.py*) für Sensor-Integration und Datenerfassung
- Datenbankfunktionen zur persistenten Speicherung der Messwerte
- Display-Steuerungsroutinen für verschiedene Anzeigeoptionen

Hardware-Komponenten:

- Raspberry Pi als zentrale Steuerungseinheit
- DHT11-Sensor für Temperatur und Luftfeuchtigkeit
- Helligkeitssensor zur Messung der Beleuchtungsstärke
- LCD-Display (2x16 Zeichen) für Textanzeige
- 7-Segment-LED-Anzeige für numerische Werte
- 8x8 LED-Matrix für Symbole (Tag/Nacht-Status)
- Relais zur Leistungssteuerung der Beleuchtung

Technische Daten

DHT11-Sensor:

- Messbereich Temperatur: 0-50°C (±2°C Genauigkeit)
- Messbereich Luftfeuchtigkeit: 20-90% RH (±5% RH Genauigkeit)
- Messintervall: 3 Sekunden

Helligkeitssensor:

- Empfindlichkeitsbereich: 0-65535 (umgerechnet in Lux)
- Schwellenwert für Tag/Nacht-Erkennung: 100 Lux

Display-Module:

- LCD: 16x2 Zeichen, I2C-Schnittstelle (Adresse 0x21)
- 7-Segment: 4-stellig, I2C-Schnittstelle (Adresse 0x70)
- LED-Matrix: 8x8 Pixel, SPI-Schnittstelle

Installation

Um das Gewächshaus-Projekt zu installieren, folgen Sie diesen Schritten:

1. Klonen Sie das Repository:

```
git clone https://github.com/vertok/Greenhouse.git
```

2. Installieren Sie die erforderlichen Pakete:

```
pip install -r requirements.txt
```

3. Führen Sie das Hauptprogramm aus:

```
python3 -m greenhouse.main --iterations 10 --interval 3
```

Beispiel für die Verwendung

Hier ist ein schnelles Beispiel, wie Sie den *ColoredLogger* aus dem *school_logging* Modul verwenden können:

```
from school_logging.log import ColoredLogger

log = ColoredLogger(__name__)
log.info("Dies ist eine Informationsnachricht.")
```

Und hier ist ein Beispiel für die Verwendung der *MeasurementSystem* -Klasse:

```
from school_logging.log import ColoredLogger
from greenhouse.messdaten import MeasurementSystem

log = ColoredLogger("gewächshaus")
messsystem = MeasurementSystem(log)

# Datenbank erstellen und initialisieren
messsystem.create_database()

# Beispielwerte zur Anzeige
temperatur = 23.5
luftfeuchtigkeit = 45.2
helligkeit = 256

# Alle Anzeigen aktualisieren
messsystem.update_all_displays(temperatur, luftfeuchtigkeit, helligke
it)
```

Anforderungen an die Lichtsteuerung

Für den optimalen Anbau von Hanfblüten ist eine präzise Lichtsteuerung essentiell. Das System überwacht:

- Die Beleuchtungsstärke in Lux für optimales Pflanzenwachstum
- Tag/Nacht-Zyklen mittels integriertem Zeitserver (NTP)
- Automatische Anpassung der Beleuchtung basierend auf Umgebungslicht und Tageszeit

Die Helligkeitswerte werden auf der LED-Matrix durch verschiedene Symbole visualisiert, um den Mitarbeitern der Gärtnerei eine schnelle Statusüberprüfung zu ermöglichen.

Matrix-Symbole:

- ☀ **Sonnensymbol** : Zeigt ausreichende Helligkeit an (>100 Lux)
- ☽ **Mondsymbol** : Zeigt niedrige Helligkeit an (<100 Lux)

MeasurementSystem

Messdaten-Modul für das Gewächshaus-Steuerungssystem

Dieses Modul stellt die Kernfunktionalität für das Auslesen von Sensordaten, die Datenbankverwaltung und die Anzeige von Informationen bereit.

Es bietet Funktionen zum: - Erfassen von Temperatur- und Luftfeuchtigkeitswerten über den DHT11-Sensor - Messen der Umgebungshelligkeit über einen Lichtsensor - Speichern aller Messdaten in einer SQLite-Datenbank - Anzeigen der Werte auf verschiedenen Display-Typen (LCD, LED-Matrix, 7-Segment) - Verarbeiten und Auswerten der gesammelten Daten

Dieses Modul bildet das Herzstück des Gewächshaus-Überwachungssystems und übernimmt die zentrale Steuerung aller Komponenten.

class greenhouse.messdaten. ColoredLogger (*name : str* , *verbose : str = 'INFO'*) [source]

Bases: `object`

Eine benutzerdefinierte Logger-Klasse, die farbige Ausgabe bietet und das Programm bei kritischen Fehlern beendet.

Diese Klasse kapselt die Komplexität der Protokollkonfiguration und bietet eine einfache Schnittstelle für die Protokollierung von Meldungen mit verschiedenen Schweregraden. Sie richtet automatisch die Protokollierung sowohl in die Konsole (mit Farben) als auch in Dateien (ohne Farben) ein.

Attribute:

name (str): Name des Loggers, normalerweise der Modulname level (int): Numerische Protokollierungsstufe (z.B. logging.INFO) logger (logging.Logger): Das konfigurierte Logger-Objekt

Beispiel:

log = ColoredLogger('gewächshaus', 'INFO') log.info('System gestartet') log.error('Sensorfehler aufgetreten')

critical (*msg : str* , ** args : Any* , *** kwargs : Any*) → None [source]

Protokolliert eine kritische Nachricht und löst einen CriticalError aus. Das Programm wird nach dieser Nachricht automatisch beendet.

Parameters :

- msg (*str*) – Die zu protokollierende Nachricht
- *args – Zusätzliche Positionsargumente für die Formatierung
- **kwargs – Zusätzliche Schlüsselwortargumente für die Formatierung

debug (*msg : str* , ** args : Any* , *** kwargs : Any*) → None [source]

Protokolliert eine Debug-Nachricht.

Parameters :

- msg (*str*) – Die zu protokollierende Nachricht
- *args – Zusätzliche Positionsargumente für die Formatierung
- **kwargs – Zusätzliche Schlüsselwortargumente für die Formatierung

error (*msg : str* , ** args : Any* , *** kwargs : Any*) → None [source]

Protokolliert eine Fehlermeldung.

Parameters :

- **msg** (*str*) – Die zu protokollierende Nachricht
- ***args** – Zusätzliche Positionsargumente für die Formatierung
- ****kwargs** – Zusätzliche Schlüsselwortargumente für die Formatierung

info (*msg : str* , ** args : Any* , *** kwargs : Any*) → None [source]
Protokolliert eine Info-Nachricht.

Parameters :

- **msg** (*str*) – Die zu protokollierende Nachricht
- ***args** – Zusätzliche Positionsargumente für die Formatierung
- ****kwargs** – Zusätzliche Schlüsselwortargumente für die Formatierung

warning (*msg : str* , ** args : Any* , *** kwargs : Any*) → None [source]
Protokolliert eine Warnungsnachricht.

Parameters :

- **msg** (*str*) – Die zu protokollierende Nachricht
- ***args** – Zusätzliche Positionsargumente für die Formatierung
- ****kwargs** – Zusätzliche Schlüsselwortargumente für die Formatierung

class greenhouse.messdaten. MeasurementSystem (*log : ColoredLogger*) [source]
Bases: `object`

Hauptklasse zur Steuerung des Gewächshaus-Überwachungssystems.

Diese Klasse verwaltet die komplette Funktionalität des Gewächshaus-Systems: - Erfassung von Sensor-Messdaten (Temperatur, Luftfeuchtigkeit, Helligkeit) - Speicherung der Daten in einer SQLite-Datenbank - Anzeige von Informationen auf verschiedenen Display-Typen - Überwachung des Tag/Nacht-Zyklus basierend auf Helligkeitsmessungen

Die Klasse initialisiert bei ihrer Erstellung alle benötigten Hardware-Komponenten und ermöglicht die zentrale Steuerung sämtlicher Sensoren und Anzeigeelemente.

Attribute:

conn: Verbindung zur SQLite-Datenbank lcd: LCD-Display-Objekt für Textanzeige matrix: LED-Matrix-Display für grafische Symbole seven_segment: 7-Segment-Display für numerische Werte brightness_channel: Kanal für Helligkeitsmessungen temperature: Aktuelle Temperatur humidity: Aktuelle Luftfeuchtigkeit

DATABASE_FILE *: str = 'greenhouse.db'*
DHT11_PIN *= 4*
LCD_COLUMNS *= 16*
LCD_I2C_ADDRESS *= 33*
LCD_ROWS *= 2*
NUM_ITERATIONS *= 3*
SEVEN_SEGMENT_I2C_ADDRESS *= 112*
TIME_SERVER *: str = '10.254.5.115'*

close_connection ( ) → None [source]

Schließt die Datenbankverbindung.

connect_to_database ( ) → None [source]

Stellt eine Verbindung zur SQLite-Datenbank her.

create_database ( ) → None [source]

Erstellt die Datenbanktabelle, falls sie nicht existiert.

display_brightness_symbol (*brightness*) [source]

Zeigt je nach Helligkeitsstufe entweder ein Tages- oder Nachtsymbol auf dem Matrix-Display an. Das Symbol bleibt sichtbar bis zum nächsten Aufruf.

Parameters :

brightness (*int*) – Der Helligkeitswert vom Sensor.

display_measurements_on_seven_segment (*temperature* , *humidity*) [source]

Zeigt Temperatur und Luftfeuchtigkeit abwechselnd auf dem 7-Segment-Display an. Zeigt Temperatur mit ‘C’-Suffix, dann Luftfeuchtigkeit mit ‘%’-Suffix.

Parameters :

- temperature (*float*) – Die gemessene Temperatur.
- humidity (*float*) – Die gemessene Luftfeuchtigkeit.

display_on_lcd (*temperature* , *humidity*) [source]

Zeigt Temperatur und Luftfeuchtigkeit auf dem LCD an.

Parameters :

- **temperature** (*float*) – Die anzuzeigende Temperatur.
- **humidity** (*float*) – Die anzuzeigende Luftfeuchtigkeit.

get_ntp_time (*ip_address : str*) → str | None [source]

Ruft die Serverzeit vom angegebenen NTP-Server ab und konvertiert sie in die lokale Zeitzone.

Parameter: ip_address (str): Die IP-Adresse des NTP-Servers.

Returns: str: Die formatierte lokale Zeit.

humidity *= 0*

initialize_brightness_sensor ( ) [source]

Initialisiert den Helligkeitssensor.

Returns :

Der initialisierte Analogeingang für den Helligkeitssensor.

Return type :

AnalogIn

initialize_lcd ( ) [source]

Initialisiert das LCD-Display.

Returns :

Das initialisierte LCD-Objekt.

Return type :

Character_LCD_I2C

initialize_matrix_display ( ) [source]

Initialisiert das 8x8 LED-Matrix-Display mit dem MAX7219-Treiber.

Returns :

Das initialisierte Matrix-Display-Objekt.

Return type :

max7219 device

initialize_seven_segment ( ) [source]

Initialisiert das 7-Segment-Display.

Returns :

Das initialisierte 7-Segment-Display-Objekt.

Return type :

Seg7x4

log *= None*

print_database ( ) → None [source]

Gibt den Inhalt der 'measurements'-Tabelle auf der Konsole aus ohne Klammern, Anführungszeichen oder andere Sonderzeichen.

read_brightness ( ) [source]

Liest den Helligkeitswert vom Sensor aus.

Returns :

Der Helligkeitswert in Lux.

Return type :

int

read_dht11_sensor (*instance* , *max_attempts = 10*) [source]

Liest Daten vom DHT11-Sensor und gibt Temperatur und Luftfeuchtigkeit zurück.

save_measurement (*temp : float* , *hum : float*) → int | None [source]

Speichert eine Messung in der Datenbank.

Parameters :

- temp (*float*) – Die gemessene Temperatur.
- hum (*float*) – Die gemessene Luftfeuchtigkeit.

temperature *= 0*

update_all_displays (*temp : float* , *hum : float* , *brightness : int*) → None [source]

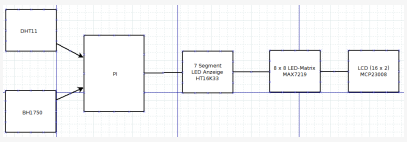
Aktualisiert alle Anzeigegeräte mit den aktuellen Messwerten.

Parameters :

- **temp** (*float*) – Die gemessene Temperatur.
- **hum** (*float*) – Die gemessene Luftfeuchtigkeit.
- **brightness** (*int*) – Die gemessene Helligkeit.

Verwendete Abbildungen

Diese Dokumentation enthält folgende Abbildungen und Diagramme:

Abbildung	Beschreibung
	Logo des Gewächshaus-Steuerungsprojekts
	Blockschaltbild des gesamten Steuerungssystems mit allen Komponenten
☀	Symbol für ausreichende Helligkeit (Tag-Modus) auf der LED-Matrix
☽	Symbol für geringe Helligkeit (Nacht-Modus) auf der LED-Matrix

API-Dokumentation

MeasurementSystem

class greenhouse.messdaten. MeasurementSystem (*log : ColoredLogger*) [source]

Bases: `object`

Hauptklasse zur Steuerung des Gewächshaus-Überwachungssystems.

Diese Klasse verwaltet die komplette Funktionalität des Gewächshaus-Systems: - Erfassung von Sensor-Messdaten (Temperatur, Luftfeuchtigkeit, Helligkeit) - Speicherung der Daten in einer SQLite-Datenbank - Anzeige von Informationen auf verschiedenen Display-Typen - Überwachung des Tag/Nacht-Zyklus basierend auf Helligkeitsmessungen

Die Klasse initialisiert bei ihrer Erstellung alle benötigten Hardware-Komponenten und ermöglicht die zentrale Steuerung sämtlicher Sensoren und Anzeigeelemente.

Attribute:

conn: Verbindung zur SQLite-Datenbank lcd: LCD-Display-Objekt für Textanzeige matrix: LED-Matrix-Display für grafische Symbole seven_segment: 7-Segment-Display für numerische Werte brightness_channel: Kanal für Helligkeitsmessungen temperature: Aktuelle Temperatur humidity: Aktuelle Luftfeuchtigkeit

DATABASE_FILE *: str = 'greenhouse.db'*
DHT11_PIN *= 4*
LCD_COLUMNS *= 16*
LCD_I2C_ADDRESS *= 33*
LCD_ROWS *= 2*
NUM_ITERATIONS *= 3*
SEVEN_SEGMENT_I2C_ADDRESS *= 112*
TIME_SERVER *: str = '10.254.5.115'*

close_connection ( ) → None [source]

Schließt die Datenbankverbindung.

connect_to_database ( ) → None [source]

Stellt eine Verbindung zur SQLite-Datenbank her.

create_database ( ) → None [source]

Erstellt die Datenbanktabelle, falls sie nicht existiert.

display_brightness_symbol (*brightness*) [source]

Zeigt je nach Helligkeitsstufe entweder ein Tages- oder Nachtsymbol auf dem Matrix-Display an. Das Symbol bleibt sichtbar bis zum nächsten Aufruf.

Parameters :

brightness (*int*) – Der Helligkeitswert vom Sensor.

display_measurements_on_seven_segment (*temperature* , *humidity*) [source]

Zeigt Temperatur und Luftfeuchtigkeit abwechselnd auf dem 7-Segment-Display an. Zeigt Temperatur mit 'C'-Suffix, dann Luftfeuchtigkeit mit '%'-Suffix.

Parameters :

- temperature (*float*) – Die gemessene Temperatur.
- humidity (*float*) – Die gemessene Luftfeuchtigkeit.

display_on_lcd (*temperature* , *humidity*) [source]

Zeigt Temperatur und Luftfeuchtigkeit auf dem LCD an.

Parameters :

- temperature (*float*) – Die anzuzeigende Temperatur.
- humidity (*float*) – Die anzuzeigende Luftfeuchtigkeit.

get_ntp_time (*ip_address : str*) → str | None [source]

Ruft die Serverzeit vom angegebenen NTP-Server ab und konvertiert sie in die lokale Zeitzone.

Parameter: ip_address (str): Die IP-Adresse des NTP-Servers.

Returns: str: Die formatierte lokale Zeit.

humidity *= 0*

initialize_brightness_sensor ( ) [source]

Initialisiert den Helligkeitssensor.

Returns :

Der initialisierte Analogeingang für den Helligkeitssensor.

Return type :

AnalogIn

initialize_lcd ( ) [source]

Initialisiert das LCD-Display.

Returns :

Das initialisierte LCD-Objekt.

Return type :

Character_LCD_I2C

initialize_matrix_display ( ) [source]

Initialisiert das 8x8 LED-Matrix-Display mit dem MAX7219-Treiber.

Returns :

Das initialisierte Matrix-Display-Objekt.

Return type :

max7219 device

initialize_seven_segment ( ) [source]

Initialisiert das 7-Segment-Display.

Returns :

Das initialisierte 7-Segment-Display-Objekt.

Return type :

Seg7x4

log = *None*

print_database ( ) → None [source]

Gibt den Inhalt der 'measurements'-Tabelle auf der Konsole aus ohne Klammern, Anführungszeichen oder andere Sonderzeichen.

read_brightness ( ) [source]

Liest den Helligkeitswert vom Sensor aus.

Returns :

Der Helligkeitswert in Lux.

Return type :

int

read_dht11_sensor (*instance* , *max_attempts = 10*) [source]

Liest Daten vom DHT11-Sensor und gibt Temperatur und Luftfeuchtigkeit zurück.

save_measurement (*temp : float* , *hum : float*) → int | None [source]

Speichert eine Messung in der Datenbank.

Parameters :

- temp (*float*) – Die gemessene Temperatur.
- hum (*float*) – Die gemessene Luftfeuchtigkeit.

temperature *= 0*

update_all_displays (*temp : float* , *hum : float* , *brightness : int*) → None [source]

Aktualisiert alle Anzeigegeräte mit den aktuellen Messwerten.

Parameters :

- temp (*float*) – Die gemessene Temperatur.
- hum (*float*) – Die gemessene Luftfeuchtigkeit.
- brightness (*int*) – Die gemessene Helligkeit.

ColoredLogger

class school_logging.log. ColoredLogger (*name : str* , *verbose : str = 'INFO'*) [source]

Bases: `object`

Eine benutzerdefinierte Logger-Klasse, die farbige Ausgabe bietet und das Programm bei kritischen Fehlern beendet.

Diese Klasse kapselt die Komplexität der Protokollkonfiguration und bietet eine einfache Schnittstelle für die Protokollierung von Meldungen mit verschiedenen Schweregraden. Sie richtet automatisch die Protokollierung sowohl in die Konsole (mit Farben) als auch in Dateien (ohne Farben) ein.

Attribute:

name (str): Name des Loggers, normalerweise der Modulname level (int): Numerische Protokollierungsstufe (z.B. logging.INFO) logger (logging.Logger): Das konfigurierte Logger-Objekt

Beispiel:

log = ColoredLogger('gewächshaus', 'INFO') log.info('System gestartet') log.error('Sensorfehler aufgetreten')

critical (*msg : str* , ** args : Any* , *** kwargs : Any*) → None [source]

Protokolliert eine kritische Nachricht und löst einen CriticalError aus. Das Programm wird nach dieser Nachricht automatisch beendet.

Parameters :

- msg (*str*) – Die zu protokollierende Nachricht
- *args – Zusätzliche Positionsargumente für die Formatierung
- **kwargs – Zusätzliche Schlüsselwortargumente für die Formatierung

debug (*msg : str* , ** args : Any* , *** kwargs : Any*) → None [source]

Protokolliert eine Debug-Nachricht.

Parameters :

- msg (*str*) – Die zu protokollierende Nachricht

- ***args** – Zusätzliche Positionsargumente für die Formatierung
- ****kwargs** – Zusätzliche Schlüsselwortargumente für die Formatierung

error (*msg : str* , ** args : Any* , *** kwargs : Any*) → None [source]

Protokolliert eine Fehlermeldung.

Parameters :

- **msg** (*str*) – Die zu protokollierende Nachricht
- ***args** – Zusätzliche Positionsargumente für die Formatierung
- ****kwargs** – Zusätzliche Schlüsselwortargumente für die Formatierung

info (*msg : str* , ** args : Any* , *** kwargs : Any*) → None [source]

Protokolliert eine Info-Nachricht.

Parameters :

- **msg** (*str*) – Die zu protokollierende Nachricht
- ***args** – Zusätzliche Positionsargumente für die Formatierung
- ****kwargs** – Zusätzliche Schlüsselwortargumente für die Formatierung

warning (*msg : str* , ** args : Any* , *** kwargs : Any*) → None [source]

Protokolliert eine Warnungsnachricht.

Parameters :

- **msg** (*str*) – Die zu protokollierende Nachricht
- ***args** – Zusätzliche Positionsargumente für die Formatierung
- ****kwargs** – Zusätzliche Schlüsselwortargumente für die Formatierung