

[zurück](#)

Python Teil 3: Funktionen und saubere Programmstruktur

Jetzt machen wir aus dem „funktioniert“-Script ein „wartbar“-Script. Traditionell gesagt: Wir ziehen die Kabel in Kanäle statt sie lose hinterm Rack baumeln zu lassen.

1) Warum Funktionen?

Problem ohne Funktionen: Alles steht in einer langen Datei, Menü und Logik vermischen sich.

Lösung: Jede Aufgabe bekommt eine eigene Funktion:

- Hosts anzeigen
- Anzahl anzeigen
- TLD filtern
- Sortieren
- (später) Export / Löschen / Ping

Das macht Code:

- leichter zu lesen
- leichter zu testen
- leichter zu erweitern

2) Dein nächstes Script: `03_funktionen.py`

Kopiere das komplett als neue Datei:

[03_funktionen.py](#)

```
#!/usr/bin/env python3
# 03_funktionen.py
# Menü-Programm mit Funktionen: Hosts sammeln, Duplikate verhindern,
# anzeigen, zählen, filtern, sortieren

# Importieren von Modulen
import os

...
Funktionen
```

Alle Funktionen haben einen Docstring, damit man später weiß, was sie tun und welche Parameter sie erwarten.
Alle Funktionen haben Typ-Hinweise, damit man später weiß, welche Datentypen sie erwarten und zurückgeben (auch wenn Python das nicht erzwingt).
Alle Funktionen haben sprechende Namen, damit man später weiß, was sie tun, ohne den Code lesen zu müssen.
Alle Funktionen haben eine klare Trennung von Ein- und Ausgabe, damit man sie später leichter testen und wiederverwenden kann.
Alle Funktionen haben eine klare Trennung von Logik und Präsentation, damit man sie später leichter anpassen und erweitern kann.

```
'''  
  
# Funktion zum Normalisieren von Text (z.B. für Vergleiche)  
def normalize(text: str) -> str:  
    """Trimmt Leerzeichen und macht alles klein (für Vergleiche)."""  
    return text.strip().lower()  
  
# Funktion zum Bildschirm löschen (plattformunabhängig)  
def cls() -> None:  
    os.system("cls" if os.name == "nt" else "clear")  
  
# Funktion zum Sammeln von Hostnamen  
def eingaben_sammeln() -> list[str]:  
    """Fragt Hosts ab und gibt eine Liste zurück (Duplikate werden  
ignoriert)."""  
    hosts: list[str] = []  
    hosts_norm: set[str] = set()  
  
    print("Gib Hostnamen ein (leer lassen zum Beenden)")  
    while True:  
        eingabe = input("Host: ")  
        if eingabe == "":  
            break  
  
        key = normalize(eingabe)  
        if key in hosts_norm:  
            print("Host bereits vorhanden.")  
            continue  
  
        hosts.append(eingabe.strip())  
        hosts_norm.add(key)  
  
    return hosts  
  
# Funktion zum Anzeigen der gesammelten Hosts  
def hosts_anzeigen(hosts: list[str]) -> None:  
    cls()  
    print("\n--- Gespeicherte Hosts ---")
```

```
if not hosts:
    print("(keine Hosts gespeichert)")
    return
for index, host in enumerate(hosts, start=1):
    print(f"{index} - {host}")

# Funktion zum Anzeigen der Anzahl der gesammelten Hosts
def anzahl_anzeigen(hosts: list[str]) -> None:
    cls()
    print(f"\nAnzahl der Hosts: {len(hosts)}")
    if not hosts:
        print("Keine Hosts eingegeben.")

# Funktion zum Filtern der Hosts nach TLD
def tld_filtern(hosts: list[str]) -> None:
    cls()
    if not hosts:
        print("Keine Hosts gespeichert.")
        return

    tld = input("Gib die TLD ein (z.B. .com, .de oder de): ").strip()
    if tld and not tld.startswith("."):
        tld = "." + tld

    treffer = [h for h in hosts if
normalize(h).endswith(normalize(tld))]
    print(f"\nHosts mit der TLD '{tld}':")
    for index, host in enumerate(treffer, start=1):
        print(f"{index} - {host}")
    print(f"\nTreffer: {len(treffer)}")

# Funktion zum Sortieren der Hosts
def sortieren(hosts: list[str]) -> None:
    cls()
    if not hosts:
        print("Keine Hosts gespeichert.")
        return

    sortierte = sorted(hosts, key=lambda x: normalize(x))
    print("\n--- Gespeicherte Hosts (sortiert) ---")
    for index, host in enumerate(sortierte, start=1):
        print(f"{index} - {host}")

# Funktion zum Anzeigen des Menüs und Abfrage der Auswahl
def menue() -> str:
    print(
        "\n1. Alle Hosts anzeigen"
        "\n2. Anzahl der Hosts anzeigen"
        "\n3. TLD filtern"
        "\n4. Sortieren"
        "\n5. Beenden"
```

```
)
    return input("Wähle eine Option: ").strip()

# Hauptfunktion, die den Ablauf steuert
def main() -> None:
    hosts = eingaben_sammeln()

    auswahl = input("\nMöchtest du die gespeicherten Hosts anzeigen?
(j/n): ")
    if normalize(auswahl) != "j":
        print("Programm wird beendet.")
        input("\nEnter zum Beenden...")
        return

    cls()

    while True:
        wahl = menu()

        if wahl == "1":
            hosts_anzeigen(hosts)
        elif wahl == "2":
            anzahl_anzeigen(hosts)
        elif wahl == "3":
            tld_filtern(hosts)
        elif wahl == "4":
            sortieren(hosts)
        elif wahl == "5":
            print("Programm wird beendet.")
            break
        else:
            print("Ungültige Option. Bitte wähle 1-5.")

    input("\nEnter zum Beenden...")

# Nur ausführen, wenn das Skript direkt gestartet wird (nicht
importiert)
if __name__ == "__main__":
    main()
```

Wichtiges neues Konzept: `if __name__ == "__main__":` sorgt dafür, dass `main()` nur startet, wenn du die Datei direkt ausführst.

3) Mini-Übung (Baustein innerhalb Baustein)

Füge eine **neue Menüoption 6** hinzu: „Host hinzufügen (nachträglich)“.

Regeln:

- Duplikate sollen weiterhin verhindert werden
- Normalisierung berücksichtigen

Hinweis: Damit das sauber geht, brauchen wir zusätzlich eine Datenstruktur für `hosts_norm` oder wir normalisieren beim Nachtragen auch gegen die Liste. (Wir machen es gleich sauber.)

4) DokuWiki-Seite (Teil 3)

DokuWiki - Python Teil 3: Funktionen und main()

Ziel: Code in kleine Bausteine zerlegen, damit er übersichtlich, wartbar und erweiterbar wird.

1. Was ist eine Funktion?

Eine Funktion ist ein benannter Codeblock.

```
def begruessung(name):  
    print("Hallo", name)
```

- `def` startet eine Funktion
 - in Klammern stehen Parameter (Eingaben)
 - `return` (optional) gibt etwas zurück
-

2. Warum Funktionen?

Ohne Funktionen wird ein Programm schnell unübersichtlich. Mit Funktionen kann man:

- Code wiederverwenden
- leichter testen
- leichter erweitern
- Fehler schneller finden

3. Beispiel: Hosts anzeigen als Funktion

```
def hosts_anzeigen(hosts):  
    for index, host in enumerate(hosts, start=1):  
        print(index, "-", host)
```

Aufruf:

```
hosts_anzeigen(hosts)
```

4. main() - der Startpunkt

Viele Programme haben eine Funktion `main()`, die alles steuert.

```
def main():  
    print("Programm startet")
```

5. Warum `if __name__ == "__main__":`?

```
if __name__ == "__main__":  
    main()
```

Das bedeutet:

- Wenn die Datei direkt gestartet wird → `main()` wird ausgeführt.
- Wenn die Datei als Modul importiert wird → `main()` wird NICHT automatisch ausgeführt.

Das ist Standard in Python.

6. Programmstruktur (überblick)

- `normalize()` und `cls()` sind Hilfsfunktionen
- `eingaben_sammeln()` sammelt Hosts und gibt eine Liste zurück
- Menüoptionen sind jeweils eigene Funktionen
- `main()` steuert den Ablauf

7. Übungsaufgaben

Übung A: Menüoption „Host hinzufügen“

Erweitere das Menü um eine Option, um nachträglich einen Host hinzuzufügen (Duplikate verhindern).

Übung B: Menüoption „Host löschen“

Erweitere das Menü um „Host löschen“ (mit Fehlerbehandlung).

Übung C: Export in Datei

Speichere die Hostliste in eine Textdatei `hosts.txt`.

Wenn du Baustein 3 ausführst und mir sagst „läuft“, machen wir als nächstes Baustein 4: **Dateien lesen/schreiben** (Hostliste speichern & wieder laden). Das ist dann die Brücke zu echten Admin-Tools.

From:
<http://wiki.nctl.de/dokuwiki/> - **Veni. Vidi. sudo rm -rf / vici.**

Permanent link:
http://wiki.nctl.de/dokuwiki/doku.php?id=python:grundlagen:03_funktionen&rev=1772290940

Last update: **28.02.2026 16:02**

