Ja, jetzt verstehe ich genau, was du meinst: Du möchtest wissen, **wie man in Python-Docstrings durch bestimmte Formatierungen und Strukturierungen** dafür sorgt, dass Tools wie IDEs (z.B. VSCode, PyCharm) oder interaktive Umgebungen beim **Hover oder Autocomplete** eine übersichtliche, gut strukturierte und gut lesbare Dokumentation anzeigen.

Du willst also **praktische Tricks und Regeln**, wie du Docstrings aufbaust, damit die IDE oder der Help-Viewer eine perfekte, gegliederte Übersicht mit Überschriften, Unterpunkten und klaren Parametern zeigt.

So strukturierst du Docstrings, damit IDEs und Tools perfekte Hovern-Dokumentation liefern

---

1. Verwende ein einheitliches Docstring-Format

Die meisten IDEs und Dokumentations-Tools unterstützen besonders gut:

- **Google Style Docstrings**
- **NumPy Style Docstrings**
- **reStructuredText (reST) Style**

Die IDEs erkennen diese Formate und formatieren beim Hover entsprechend sauber.

Mein Tipp: Google Style ist am einfachsten und wird von den meisten Tools gut erkannt.

---

2. Überschriften in Docstrings für Parameter, Rückgabe und Fehler

Wie du Überschriften machst:

- Schreibe klare Abschnittstitel **ganz links, gefolgt von einem Doppelpunkt**.
- Beispiel:

```
Args:
Returns:
Raises:
Examples:
```

- Diese werden von IDEs als Überschriften erkannt und gruppieren darunter die jeweiligen Beschreibungen.

---

3. Parameter sauber als Unterpunkte schreiben

Damit die Parameter im Hover sauber als Liste erscheinen:

- Schreibe jeden Parameter in einer eigenen Zeile unter `Args:` (oder `Parameters:`).
- Gib den Parameternamen, den Typ in Klammern und dann eine kurze Beschreibung an.
- **Einrückung (meist 4 Leerzeichen)** für alle Parameterbeschreibungen ist wichtig.

Beispiel:

```
Args:
    x (int): Beschreibung von x.
    y (str): Beschreibung von y.
```

Viele IDEs zeigen dann beim Hover eine saubere Tabelle oder Liste der Parameter mit Typ und Beschreibung.

---

4. Mehrzeilige Parameterbeschreibungen und Unterpunkte

Wenn du für einen Parameter noch Details oder Unterpunkte angeben willst, nutze **weiter Einrückungen**.

Zum Beispiel:

```
Args:
    config (dict): Konfigurationsoptionen mit folgenden Schlüsseln:
        - 'host' (str): Hostname.
        - 'port' (int): Portnummer.
```

So werden die Unterpunkte im Hover als Unterliste angezeigt und die Übersicht bleibt erhalten.

---

5. Rückgabewert formatieren

- Schreibe eine `Returns:`-Sektion mit Typ und kurzer Beschreibung.
- Auch hier idealerweise eine Zeile pro Rückgabewert.

```
Returns:
    bool: True, wenn erfolgreich.
```

---

6. Fehler (Exceptions) angeben

- Schreibe eine `Raises:`-Sektion.
- Liste mögliche Exceptions mit Erklärung.

```
Raises:
    ValueError: Wenn der Eingabewert ungültig ist.
```

---

7. Beispiele für besseren Kontext

- Schreibe eine `Examples:`-Sektion.
- Schreibe typische Anwendungsbeispiele in der Form von Python-Interpretersitzungen (`>>>`).

```
Examples:
    >>> meine_funktion(5, "Test")
    True
```

Viele IDEs zeigen diesen Bereich beim Hover in einer separaten Box oder einem extra Fenster an.

---

8. Ein vollständiges Minimal-Beispiel, das perfekt im Hover erscheint

```
def meine_funktion(x, y, config=None):
    """
    Verarbeitet Eingabewerte und gibt True zurück, wenn erfolgreich.

    Args:
        x (int): Erste Zahl.
        y (str): Textbeschreibung.
        config (dict, optional): Konfigurationsparameter mit:
            - 'verbose' (bool): Ausgabe aktivieren.
            - 'timeout' (int): Timeout in Sekunden.

    Returns:
        bool: True bei Erfolg, sonst False.

    Raises:
        ValueError: Wenn x negativ ist.

    Examples:
        >>> meine_funktion(5, "Hallo")
        True
    """
```

---

9. Warum funktionieren diese Tricks?

- IDEs und Tools parsen den Docstring nach bestimmten **Schlüsselwörtern und Formatierungen**.
- Einrückungen signalisieren Unterpunkte.
- Doppelpunkte markieren Abschnittsüberschriften.
- Einheitliches Format ermöglicht eine strukturierte Darstellung statt eines unübersichtlichen Fließtexts.

Zusammenfassung: So erzielst du perfekte Hover-Dokumentation

Was tun?	Wie genau?	Effekt im Hover
Überschriften setzen	`Args:`, `Returns:`, `Raises:` jeweils am Zeilenanfang mit `:`	Klar erkennbare Abschnitte
Parameter einzeln auflisten	`param_name (Typ): Beschreibung` mit 4 Leerzeichen eingerückt	Übersichtliche Parameterliste
Unterpunkte einrücken	Weitere 4 Leerzeichen + `-` oder `*` für Listenpunkte	Strukturierte, verschachtelte Details
Beispiele mit `>>>`-Syntax	Beispielcode in `Examples:` Abschnitt	Anschauliche Anwendungsbeispiele
Fehler dokumentieren	`Raises:` mit Exception und Grund	Hinweise auf mögliche Fehler

---

Wenn du diese einfachen Regeln befolgst, sieht deine Docstring-Dokumentation in fast jeder IDE beim Hover perfekt aus und hilft dir und anderen, Funktionen sofort richtig zu verstehen.

Wenn du möchtest, kann ich dir noch genau erklären, wie du z.B. mit Sphinx oder anderen Tools aus diesen Docstrings sogar Webseiten oder PDFs machst. Möchtest du das?