Klar, hier kommt dein ausführlicher Lernzettel zu **`argparse`** in Python – perfekt für dein Linux-Terminalprogramm mit Bash-Skripten, mit vielen Erklärungen und Beispielcode:

---

# Lernzettel: Python-Modul `argparse`

---

## 1. Was ist `argparse`?

- `argparse` ist ein Standardmodul, um Kommandozeilenargumente (Parameter, Optionen) für dein Programm zu definieren und auszuwerten.
- Es hilft dir, Eingaben wie `python meinprog.py --datei test.sh -v` einfach zu verarbeiten.
- Du bekommst automatisch generierte Hilfe (`-h` / `--help`) und Fehlerprüfungen.

---

## 2. Warum `argparse` verwenden?

- Terminalprogramme brauchen oft Parameter: Dateipfade, Flags, Einstellungen.
- `argparse` macht das Handling einfach, lesbar und sicher.
- Es übernimmt Validierung, Default-Werte und Fehlerbehandlung.
- Für Nutzer: Klar verständliche Hilfetexte.

---

## 3. Grundstruktur eines `argparse`-Programms

```
import argparse

parser = argparse.ArgumentParser(description="Mein Programm zur Ausführung von
Bash-Skripten")

# Argumente definieren
parser.add_argument("script", help="Pfad zum Bash-Skript, das ausgeführt werden
soll")

# Optionale Flags
parser.add_argument("-v", "--verbose", action="store_true", help="Ausführliche
Ausgabe aktivieren")
parser.add_argument("-t", "--timeout", type=int, default=10, help="Maximale
Ausführungszeit in Sekunden")

# Argumente parsen
args = parser.parse_args()

print(f"Starte Skript: {args.script}")
if args.verbose:
    print("Verbose-Modus aktiviert")
print(f"Timeout gesetzt auf: {args.timeout} Sekunden")
```

**Erläuterung:**

- `ArgumentParser` erstellt den Parser.
- `add_argument` definiert Parameter (Pflicht und optional).
- `action="store_true"` macht aus einem Flag einen booleschen Wert (True wenn gesetzt).
- `type=int` wandelt den Parameter in einen Integer um.
- `default` definiert Standardwert, falls der Parameter nicht gesetzt wird.

---

## 4. Positionale vs optionale Argumente

- **Positionale Argumente** müssen immer angegeben werden, und zwar in der Reihenfolge. Beispiel: `"script"` oben ist ein solcher Parameter.
- **Optionale Argumente** sind Flags oder Optionen mit `-` oder `--` (z.B. `-v` oder `--timeout`), die man setzen kann oder auch nicht.

---

## 5. Argumenttypen und Wertevalidierung

- `type` gibt an, wie das Argument umgewandelt werden soll (z.B. `int`, `float`, `str`).
- Beispiel:

```
parser.add_argument("--count", type=int, help="Anzahl der Ausführungen")
```

- Wenn ein Nutzer einen falschen Typ eingibt (z.B. `--count abc`), zeigt `argparse` automatisch einen Fehler.

---

## 6. Flags und Boolean-Werte

- Für simple An/Aus-Optionen benutzt du `action="store_true"` oder `store_false`:

```
parser.add_argument("-d", "--debug", action="store_true", help="Debug-Modus
einschalten")
```

- Wenn Nutzer `-d` oder `--debug` schreiben, wird `args.debug` zu `True`, sonst `False`.

---

## 7. Standardwerte & Pflichtangaben

- Mit `default` kannst du Standardwerte definieren, falls ein Argument nicht angegeben wird.
- Mit `required=True` machst du ein optionales Argument verpflichtend (z.B. bei `--config`):

```
parser.add_argument("--config", required=True, help="Pfad zur
Konfigurationsdatei")
```

---

## 8. Mehrere Werte / Listen einlesen

- Mit `nargs` kannst du angeben, wie viele Werte ein Argument annimmt:

```
parser.add_argument("--files", nargs="+", help="Liste von Skripten, die
ausgeführt werden")
```

- `nargs="+"` bedeutet: mindestens ein Wert, beliebig viele möglich.
- So kann der Nutzer z.B. `--files script1.sh script2.sh script3.sh` eingeben.

---

## 9. Beispiel für dein Bash-Skript-Runner-Programm

```
import argparse
import subprocess

parser = argparse.ArgumentParser(description="Starte Bash-Skripte mit Optionen")

parser.add_argument("scripts", nargs="+", help="Pfad(e) zu Bash-Skript(en), die
ausgeführt werden")
parser.add_argument("-v", "--verbose", action="store_true", help="Ausführliche
Ausgabe")
parser.add_argument("-t", "--timeout", type=int, default=10, help="Timeout in
Sekunden")

args = parser.parse_args()

for script in args.scripts:
    if args.verbose:
        print(f"Starte Skript: {script} mit Timeout {args.timeout}s")
    try:
        result = subprocess.run([script], capture_output=True, text=True,
timeout=args.timeout)
        print(f"Rückgabecode: {result.returncode}")
        if args.verbose:
            print("Standardausgabe:")
            print(result.stdout)
            if result.stderr:
                print("Fehlerausgabe:")
                print(result.stderr)
    except subprocess.TimeoutExpired:
        print(f"Timeout: Skript {script} wurde abgebrochen!")
```

---

## 10. Hilfe automatisch erzeugen

- `argparse` generiert automatisch eine Hilfe mit `-h` oder `--help`
- Beispiel:

```
python meinprog.py -h
```

gibt sowas aus:

```
usage: meinprog.py [-h] [-v] [-t TIMEOUT] scripts [scripts ...]

Starte Bash-Skripte mit Optionen

positional arguments:
  scripts               Pfad(e) zu Bash-Skript(en), die ausgeführt werden

optional arguments:
  -h, --help            show this help message and exit
  -v, --verbose         Ausführliche Ausgabe
  -t TIMEOUT, --timeout TIMEOUT
                        Timeout in Sekunden (default: 10)
```

---

## 11. Tipps für dein Programm

- Nutze **Positionsargumente** für wichtige Daten, z.B. welche Skripte laufen sollen.
- Nutze **optionale Argumente** für Flags und Einstellungen, z.B. Timeout, verbose, Debug.
- Verwende `type` und `default`, damit dein Programm robust ist.
- Prüfe bei komplexen Eingaben `nargs` (z.B. mehrere Skripte).
- Lass `argparse` die Fehler- und Hilfeseite generieren, spart dir viel Arbeit.

---

## 12. Beispiel: Mehrere optionale Flags mit verschiedenen Aktionen

```
parser.add_argument("-q", "--quiet", action="store_true", help="Keine Ausgabe
außer Fehlern")
parser.add_argument("-r", "--retry", type=int, default=1, help="Anzahl der
Wiederholungen bei Fehler")
```

- `quiet` ist ein boolsches Flag (True wenn gesetzt).
- `retry` ist eine Zahl, mit der du eine Funktion z.B. mehrfach ausführen kannst.

---

## 13. Fazit

- `argparse` macht es dir leicht, komplexe Kommandozeilenparameter verständlich und sicher zu verarbeiten.
- Dein Terminalprogramm wird dadurch professioneller und nutzerfreundlicher.
- Es lohnt sich, zu Beginn deine Argumente gut zu planen und ordentlich zu beschreiben.

---

Wenn du möchtest, kann ich dir auch Beispiele zeigen, wie man mit `argparse` interaktive Menüs baut oder dynamisch Parameter prüft.
Oder wir bauen zusammen dein konkretes Skript-Launcher-Programm mit allen gewünschten Optionen!