Variabelen en datatypes: In Python worden variabelen dynamisch getypeerd; je schrijft simpelweg `naam = waarde` (zonder type-aanduiding). Veelgebruikte types zijn `int`, `float`, `str`, `list`, `dict` enz. String-literals staan tussen aanhalingstekens en lijsten tussen blokhaken, bijvoorbeeld `mijn_list = []` voor een lege lijst. Indentatie (inspringen) bepaalt de scope van codeblokken. Een voorbeeld van een stringformattering is een *f-string*:

```
naam = "Jan"
print(f"Hallo {naam}!")  # toont: Hallo Jan!
```

F-strings maken het eenvoudig om variabelen in tekst te verwerken [1].

- **String-methode `split` en `join`:** De methode `str.split(delimiter)` splitst een string op een opgegeven scheidingsteken en geeft een lijst terug, terwijl `delimiter.join(list)` een lijst van strings samenvoegt tot één string [2]. Bijvoorbeeld:

  ```
  s = "appel,peer,banaan"
  parts = s.split(",")        # ['appel', 'peer', 'banaan']
  result = ";".join(parts)    # 'appel;peer;banaan'
  ```

  Dit is handig voor het inlezen of opslaan van CSV-achtige tekst [2].

- **Lijsten:** Gebruik lijsten om verzamelingen van items op te slaan. Met `lijst.append(x)` voeg je een item toe aan het einde van de lijst [3]. Met `lijst.pop(i)` verwijder je het element op index `i` (zonder argument verwijdert en retourneert `pop()` het laatste element) [4]. Enkele voorbeelden:

  ```
  fruits = ['appel', 'peer']
  fruits.append('banaan')    # fruits is nu ['appel', 'peer', 'banaan'] [3]
  laatste = fruits.pop()     # verwijdert 'banaan' en returnt het [4]
  ```

- **User input en output:** Met de functie `input(prompt)` pauzeert het programma en wacht op gebruikersinvoer als string. Gebruik `print()` om informatie te tonen in de console. Bijvoorbeeld:

```
naam = input("Wat is uw naam? ")   # leest gebruikersnaam als string
leeftijd = int(input("Leeftijd: "))  # int() converteert naar geheel getal
print(f"Welkom, {naam}! U bent {leeftijd} jaar oud.")
```

`input()` geeft altijd een string terug, zelfs als de gebruiker cijfers typt [5]. Gebruik dus `int()` of `float()` om numerieke invoer te verwerken. Beide functies zijn essentieel voor console-interactie [6] [5].

Lussen (loops)

In Python zijn er twee hoofdtypen lussen:

- `for`**-lus:** Hiermee herhaal je een blok code voor elk element in een iterabele (zoals een lijst of string) [7]. Bijvoorbeeld:

```
for boek in boekenlijst:
    print(boek)
```

Hier wordt `print(boek)` uitgevoerd voor elk item in `boekenlijst` [7]. Met `range(n)` kun je een sequentie getallen genereren, bv. `for i in range(5):`.

- `while`**-lus:** Voert herhaald een codeblok uit zolang een conditie waar (`True`) is [7]. Bijvoorbeeld:

```
pogingen = 0
while pogingen < 3:
    print("Poging nummer", pogingen+1)
    pogingen += 1
```

Dit blijft lopen tot `pogingen` niet langer `< 3` is [7].

In beide type lussen kun je de statements `break` en `continue` gebruiken (binnen een `if`-voorwaarde) voor extra controle:
- `break` beëindigt onmiddellijk de lus en gaat verder na de lus [8].
- `continue` slaat de rest van de huidige iteratie over en begint de volgende iteratie van de lus [8].
Bijvoorbeeld:

```
for i in range(10):
    if i == 5:
        break     # stopt de lus zodra i gelijk is aan 5 [8]
    print(i)
```

Beslissingen (if/elif/else)

Voor conditionele logica gebruik je `if`, eventueel gevolgd door `elif` (else-if) en een optionele `else`. De blokken moeten door inspringing gemarkeerd worden. Bijvoorbeeld:

```
if gebruikersnaam == "admin":
    print("Welkom, beheerder!")
elif gebruikersnaam == "":
    print("Geen naam ingevuld.")
else:
    print("Welkom, gewone gebruiker!")
```

- De `if`-voorwaarde (zoals `gebruikersnaam == "admin"`) wordt getest; als die `True` is, wordt alleen het eerste blok uitgevoerd.
- Een `elif`-voorwaarde wordt alleen getest als de vorige `if` of `elif` niet waar was.
- Het optionele `else`-blok draait als geen van de voorgaande condities waar was.
Volgens de syntax: `if: ... elif: ... else: ...` [9] [10].

Een voorbeeld met getallen:

```
x = int(input("Geef een getal: "))
if x > 0:
    print("Positief getal")
elif x < 0:
    print("Negatief getal")
else:
    print("Nul")
```

Hier worden meerdere condities afgehandeld met `if…elif…else` [10].

Functies

Met functies kun je herbruikbare codeblokken definiëren. Een functie maak je met het sleutelwoord `def`:

```
def mijn_functie(parameter1, parameter2):
    # codeblokken
    resultaat = parameter1 + parameter2
    return resultaat
```

Deze functie heet `mijn_functie` en neemt twee parameters. Je roept een functie aan door `mijn_functie(arg1, arg2)` te schrijven. Bijvoorbeeld:

```python
def groet(naam):
    print(f"Hallo, {naam}!")

groet("Piet")  # roept de functie aan; toont "Hallo, Piet!"
```

Zoals W3Schools uitlegt, wordt de code in een functie pas uitgevoerd wanneer je de functie **aanroept** (met naam en haakjes) [11]. Functies maken je code overzichtelijk en herbruikbaar.

CSV-bestanden

CSV-bestanden (Comma-Separated Values) zijn tekstbestanden met gescheiden kolommen. Python heeft de ingebouwde `csv`-module om dergelijke bestanden eenvoudig te lezen en te schrijven [12] [13].

- **Lezen:**

```python
import csv
with open('bijlage3.csv', mode='r', newline='') as bestand:
    reader = csv.reader(bestand, delimiter=';')
# delimiter kan ',' of ';' zijn [14]
    for rij in reader:
        print(rij)
```

Hierbij geeft `csv.reader` per regel een lijst terug met kolomwaarden (als strings) [15] [16]. Bijvoorbeeld: elke `rij` is een Python-lijst zoals `['1', 'Alice', 'Cook', '32']` [15]. Let op: de standaard-`delimiter` is een komma, maar je kunt ook een puntkomma of ander teken opgeven [14].

- **Schrijven:**

```python
import csv
data = [
    ['kolom1', 'kolom2', 'kolom3'],  # bijv. koppen
    ['1', 'Alice', 'Cook'],
    ['2', 'Bob', 'Portier']
]
with open('uitvoer.csv', 'w', newline='') as bestand:
    writer = csv.writer(bestand, delimiter=';')
    writer.writerows(data)
```

Hier maakt `csv.writer` van elke sub-lijst een regel in het CSV-bestand [13]. Het argument `newline=''` voorkomt lege regels tussen de rijen in het bestand.

Gebruik deze patronen om snel CSV-data in te laden, te bewerken (bijvoorbeeld in lijsten van lijsten) en weer weg te schrijven naar het bestand.

Console-menu (voorbeeld workflow)

Om een keuzemenu in de console te maken, combineer je een oneindige `while True`-lus met een `if/elif`-afhandeling. Voorbeeld:

```
while True:
    print("=== Menu ===")
    print("1. Toevoegen")
    print("2. Verwijderen")
    print("0. Stoppen")
    keuze = int(input("Maak uw keuze: "))  # numerieke invoer via input()
    if keuze == 0:
        break                    # verlaat de lus (en het programma) [8]
    elif keuze == 1:
        # code voor optie 1
        print("Toevoegen gekozen")
    elif keuze == 2:
        # code voor optie 2
        print("Verwijderen gekozen")
    else:
        print("Ongeldige keuze, probeer opnieuw.")
print("Programma beëindigd.")
```

In dit voorbeeld wordt steeds het menu getoond en gevraagd om invoer. Met `break` stap je uit de loop [8] (bij keuze 0) en stopt het menu. Voor elke andere keuze ga je via `elif` naar de juiste code. Dit patroon zie je vaak bij een “admin-panel” of keuzemenu-systeem.

Samenvatting van handige punten

- **Lussen:** gebruik `for item in lijst:` voor iteratie of `while voorwaarde:` voor voorwaardelijke herhaling [7].
- **Voorwaarden:** gebruik `if…elif…else` om beslissingen te nemen [9] [10].
- **Functies:** definieer met `def naam():` om code te hergebruiken [11].
- **Input/print:** `input()` leest altijd een string, gebruik `int(input())` voor getallen [6] [5].
- **Lijsten:** `lijst.append(x)` voegt toe, `lijst.pop(i)` verwijdert element i [3] [4].
- **Strings:** `s.split(',')` deelt string op, `','.join(list)` voegt samen [2].
- **CSV:** gebruik de `csv`-module: `csv.reader` voor lezen (per rij een lijst) en `csv.writer` voor schrijven [16] [13].

Deze voorbeelden en uitleg beslaan veelvoorkomende bouwstenen (loops, functies, file I/O, lijsten, stringmanipulatie) die je kan gebruiken om een bestaand Python-script aan te passen zoals in een

bibliotheek-systeem. Pas deze patterns aan de hand van de opdracht aan, documenteer je wijzigingen (met commentaar in de code), en test het script met de aangeleverde CSV-data.

Bronnen: Bovenstaande informatie is gebaseerd op Python-documentatie en tutorials [7] [11] [16] [13] [8] [9] [2] [6] [3] [12].

---

[1] [2] Python String Methods, with Examples — SitePoint
https://www.sitepoint.com/python-string-methods/

[3] [4] 5. Data Structures — Python 3.13.4 documentation
https://docs.python.org/3/tutorial/datastructures.html

[5] [6] Basic Input and Output in Python – Real Python
https://realpython.com/python-input-output/

[7] Loops in Python – For, While and Nested Loops | GeeksforGeeks
https://www.geeksforgeeks.org/loops-in-python/

[8] Python break and continue (With Examples)
https://www.programiz.com/python-programming/break-continue

[9] [10] Python if, if...else Statement (With Examples)
https://www.programiz.com/python-programming/if-elif-else

[11] Python Functions
https://www.w3schools.com/python/python_functions.asp

[12] [14] [15] CSV lezen met Python (Dutch) - DEV Community
https://dev.to/compilerboiler/csv-lezen-met-python-dutch-2l16

[13] [16] Reading and Writing CSV Files in Python | GeeksforGeeks
https://www.geeksforgeeks.org/reading-and-writing-csv-files-in-python/