1.8 Unterprogramme

Ein Programm aus einem Stück ist ab etwa fünfzig Zeilen nicht mehr zu überblicken. Die Gegenmaßnahme ist alt und wirksam: Man zerlegt es in benannte Teile, die man einzeln verstehen, einzeln testen und mehrfach verwenden kann.

Definition: Unterprogramm

Ein **Unterprogramm** ist eine benannte, abgeschlossene Anweisungsfolge, die von anderer Stelle aus aufgerufen werden kann. Zwei Bauarten:

- Eine **Prozedur** führt etwas aus und gibt keinen Wert zurück. Sie ist ein **Auftrag**.
- Eine **Funktion** berechnet einen Wert und gibt ihn zurück. Sie ist eine **Anfrage**.

Diese Unterscheidung ist keine Formalie. Sie taucht in Kapitel 2 unverändert wieder auf — dort heißen die Unterprogramme *Methoden*, und ein Objekt bekommt entweder einen Auftrag („ändere deine Farbe") oder eine Anfrage („welche Farbe hast du?").

Merke

Auftrag oder Anfrage? Fragen Sie: Will ich etwas *wissen* oder etwas *bewirken*? Wissen → Funktion. Bewirken → Prozedur. Ein Unterprogramm, das beides tut, ist meist eines zu wenig.

In Python gibt es nur `def`; eine Funktion ohne `return` liefert `None` und ist damit faktisch eine Prozedur. Pascal unterscheidet die beiden im Schlüsselwort.

Anfrage (Funktion)

```python
def flaeche(breite, hoehe):
    return breite * hoehe
```

```pascal
function Flaeche(breite, hoehe: Real): Real;
begin
  Result := breite * hoehe;
end;
```

Auftrag (Prozedur)

```python
def ausgeben(text):
    print(">>> " + text)
```

```pascal
procedure Ausgeben(text: String);
begin
  WriteLn('>>> ', text);
end;
```

1.8.1 Parameter: Wert oder Referenz

Beim Aufruf werden **Argumente** an die **Parameter** übergeben. Entscheidend ist, *wie*.

Definition: Wert- und Referenzparameter

Bei einem **Wertparameter** erhält das Unterprogramm eine **Kopie**. Änderungen daran bleiben drinnen.

Bei einem **Referenzparameter** erhält es die **Variable selbst**. Änderungen wirken nach außen. In Pascal schreibt man dafür `var` vor den Parameter.

Beispiel: Tauschen — einmal falsch, einmal richtig

```
1  procedure TauschFalsch(a, b: Integer);      { Wertparameter }
2  var hilf: Integer;
3  begin
4    hilf := a;  a := b;  b := hilf;
5  end;
6
7  procedure TauschRichtig(var a, b: Integer); { Referenzparameter }
8  var hilf: Integer;
9  begin
10   hilf := a;  a := b;  b := hilf;
11 end;
```

Nach `TauschFalsch(x, y)` stehen `x` und `y` unverändert da: Getauscht wurden die Kopien. Nur ein einziges Wort unterscheidet die beiden Fassungen.

Python kennt kein `var`. Dort gilt: Zahlen, Zeichenketten und Wahrheitswerte verhalten sich wie Wertparameter, Listen und Objekte wie Referenzparameter.

```
1 def haenge_an(liste):         # liste ist eine Liste
2     liste.append(99)          # wirkt nach außen
3
4 def ersetze(liste):
5     liste = [99]              # wirkt NICHT nach außen
```

Stolperstein

Im zweiten Fall zeigt der Name `liste` danach auf eine neue Liste — die alte bleibt unberührt. Verändern Sie das übergebene Objekt (`append`, `sort`, Zuweisung an `liste[i]`), oder geben Sie ein Ergebnis zurück. Eine Zuweisung an den Parameternamen selbst erreicht den Aufrufer nie.

1.8.2 Lokal und global

Eine im Unterprogramm deklarierte Variable ist **lokal**: Sie entsteht beim Aufruf und verschwindet am Ende. Eine außerhalb deklarierte Variable ist **global** und überall sichtbar.

Globale Variablen sind bequem und darum gefährlich: Wenn jedes Unterprogramm an derselben Variablen schreiben darf, lässt sich ein Fehler nicht mehr eingrenzen. Jede Stelle im Programm kommt als Verursacher in Frage.

Merke

Ein Unterprogramm soll alles, was es braucht, über seine Parameter bekommen und alles, was es liefert, über seinen Rückgabewert abgeben. Dann — und nur dann — kann man es isoliert testen.

Diese Regel ist der Kern dessen, was Kapitel 2 das **Geheimnisprinzip** nennt.

Arbeitsauftrag 4

11. Schreiben Sie eine Funktion `ggt(a, b)` nach dem euklidischen Algorithmus — in beiden Sprachen.
12. Schreiben Sie eine Prozedur, die eine Zahl in Worten ausgibt (`3` → `drei`), für 0 bis 9. Begründen Sie, warum hier eine Prozedur und keine Funktion naheliegt — und wann das Gegenteil besser wäre.
13. Was gibt das folgende Programm aus? Begründen Sie, ohne es laufen zu lassen.

```python
def f(x, liste):
    x = x + 1
    liste.append(x)
    return x

a = 5
b = []
f(a, b)
print(a, b)
```

14. Überführen Sie `TauschRichtig` nach Python. Was müssen Sie ändern — und warum?

Selbstcheck

Können Sie ohne nachzusehen sagen, welche der drei Schleifenarten mindestens einmal läuft? Und können Sie an einem eigenen Beispiel erklären, warum ein Wertparameter beim Tauschen versagt?

Gymnasium Ottweiler