Rekursion und Iteration sind zwei Konzepte in der Informatik, mit deren Hilfe wiederholte Berechnungen programmiert werden können.

**Rekursion** (lat. zurücklaufen) = eine Programmteil (Unterprogramm, Formel, Funktion) ruft sich im eigenen Code selbst auf[1]. Das klassische Beispiel für die Rekursion ist die Definition der Fakultät: n! = n * (n-1)!

Eine Rekursion ist so aufgebaut: um den Wert der Funktion zu bestimmen, müssen mehrere Vorgänger bestimmt werden, was bei jedem Aufruf der Funktion erfolgt. Die Rekursion ruft sich selbst auf, bis eine Abbruchbedingung (Base Case) erreicht ist. Eine Rekursion ist dann von Vorteil, wenn sich das Problem in Teilprobleme aufspalten lässt (z.B. bei binärer Suche Merge Sort, Turm von Hanoi). Gefahr des Stack Overflow (Pufferüberlauf)!

```
# Beispiel Fakultaet als Rekursion
def fakultaet(n):
   # Abbruchbedingung
   if n == 1:
      return 1
   # rekursive Aufruf
   else:
      return n * fakultaet(n - 1)
zahl = int(input("Bitte eine ganze Zahl eingeben: "))
print(fakultaet(zahl))
```

**Iteration** (lat. Wiederholung) Unter Iteration versteht man ein mehrmaliges Ausführen einer Aktion (eines oder mehrerer Anweisungen) mit Hilfe einer Schleife. Durch die Wiederholungen erfolgt eine schrittweise Annäherung an ein vorgegebenes Ende. Die Anzahl der Wiederholungen kann bei Bedarf mitgezählt werden. Typische Fälle: Zählen, Summenbildung, Fibonacci. Gefahr der Endlosschleife!

```
# Beispiel Fakultaet als Iteration
zahl = int(input("Bitte eine ganze Zahl eingeben: "))
n = 1
for i in range(2, zahl + 1):
   n *= i
print(n)
```

**Fazit**: Mit Rekursion kann nicht mehr berechnet werden als mit der Iteration. Alles, was mit Iteration geht, geht auch mit Rekursion und umgedreht. Iteration braucht meist weniger Speicherplatz und ist schneller. Rekursion ist nützlich bei verzweigten Problemen.

[1] vgl. Fischer, Peter; Hofer, Peter. Lexikon der Informatik, Heidelberg, London, New York: Springer, 15. Auflage, 2011.