Mit der `format()`-Methode, lassen sich unterschiedliche Datentypen wie Zeichenketten und Zahlen bei der Ausgabe flexibel formatieren. Dies betrifft:

- Das optionale Anzeigen eines positiven Vorzeichens
- Einfügen von Füllzeichen (z. B. um den Output tabellarisch auszugeben)
- Die Ausrichtung der Ausgabe (linksbündig, rechtsbündig, zentriert)
- Abtrennen von Zifferngruppen (10'000'000)
- Darstellungsgenauigkeit von Gleitkommazahlen
- Ausgabeformat ganzer Zahlen (binär, oktal, dezimal, hexadezimal, ...)

Um diese Informationen an Python zu übermitteln, muss sie in einen *Formatstring* verpackt werden. Damit Python diesen interpretieren kann, muss er – ähnlich wie ein Computerprogramm oder ein natürlichsprachlicher Satz – syntaktisch korrekt aufgebaut sein.

In unserem Fall wird der Formatausdruck durch eine *kontextfreie Grammatik* definiert und erzeugt. Ein Beispiel soll die folgende formale Beschreibung verständlicher: Man möchte Bitfolgen der Form `11`, `101`, `1001`, `10001`, ... formal beschreiben.

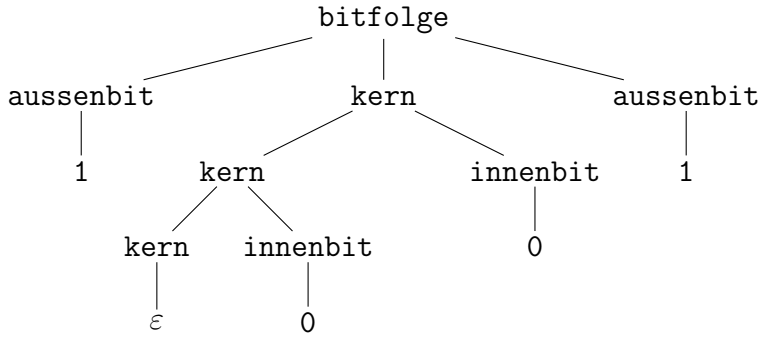
```
bitfolge   =  aussenbit kern aussenbit
aussenbit  =  "1"
kern       =  ε
kern       =  kern innenbit
innenbit   =  "0"
```

Diese Menge von *Produktionsregeln* wird so interpretiert:

- `bitfolge` besteht aus einem `aussenbit`, gefolgt von einem `kern` und gefolgt von einem weiteren `aussenbit`.
- Ein `aussenbit` besteht aus dem Zeichen `1`. Die Anführungs- und Schlusszeichen besagen, dass es sich um ein *Terminalsymbol* handelt, das im Gegensatz zu den Nichtterminalsymbolen (ohne Anführungs- und Schlusszeichen) nicht mehr weiter „zerlegt" werden kann. Die Produktionsregel endet also hier.
- Ein `kern` besteht entweder aus dem *leeren Wort* ε („empty") oder aus einem `kern`, gefolgt von einem `innenbit`.
- Ein `innenbit` ist hier das Terminalsymbol `0`.

Wenn man nun, beginnend mit dem *Startsymbol*, die Produktionsregeln in einer fest gewählten Weise anwendet, bis nur noch Terminalsymbole vorhanden sind, entsteht ein *Wort*, das den Regeln dieser Sprache genügt.

Umgekehrt kann lässt sich damit überprüfen, ob eine konkrete Bitfolge wie `1001` zu dieser Sprache gehört (formal korrekt ist), indem man einen Syntax-Baum erstellt.



Formal besteht eine kontextfreie Grammatik aus

- einer endlichen Menge T von Terminalsymbolen,
- einer endlichen Menge V von Variablen mit $T \cap V = \emptyset$,
- einer endlichen Menge von Produktionsregeln P die Variablen eine Folge von Variablen, von Terminalsymbolen oder eine Kombination aus beiden zuordnen,
- Einem Startsymbol $S \in V$.

Mit Metazeichen lässt sich die Anzahl der Produktionsregeln reduzieren:

- `"|"` (Alternative). Beispiel: `kern = "" | kern innenbit`
- `"+"` (kommt mindestens einmal vor). Beispiel:

  ```
  ziffer = "0" | "1" | "2" | ... | "9"
  zahlr = digit+
  ```

  Eine `zahl` besteht also aus mindestens einer `ziffer` (z. B. `"2019"` oder `"007"`).
- `"[...]"` (Option). Beispiel:

  `a = [b] c` ist eine Kurzschreibweise für

  ```
  a = c
  a = b c
  ```

Es gibt noch weitere solche Metazeichen. Für das Verständnis von dem, was folgt, sind sie aber nicht nötig.

Der Ausdruck der Format-Spezifikation muss gemäss Python-Dokumentation die folgende Gestalt haben:

```
format_spec = [[fill]align][sign][#][0][width][grouping_option][.precision][type]
fill            = <any character>
align           = "<" | ">" | "=" | "^"
sign            = "+" | "-" | " "
width           = digit+
grouping_option = "_" | ","
precision       = digit+
type            = "b"|"c"|"d"|"e"|"E"|"f"|"F"|"g"|"G"|"n"|"o"|"s"|"x"|"X"|"%"
```

Das Startsymbol ist `format_spec`. Terminalsymbole sind in Anführungs und Schlusszeichen eingeschlossen. Die Metazeichen `"[...]"`, `"|"` und `"+"` kommen vor. Die folgende Produktionsregeln muss man sich noch dazudenken:

```
<any character> = "a" | ... | "z" | "A" | ... | "Z" | ...
digit = "0" | "1" | ... | "9"
```

Zudem sollten in der Produktionsregel für das Startsymbol noch die Symbole `"#"` und `"0"` durch Anführungs- und Schlusszeichen als Terminalsymbole gekennzeichnet werden. Die Interpretation dieser Symbole gehört aber nicht zum Prüfungsstoff.

Beispiele:

- `"{0:*>10}".format("abc")`

  Stelle den String `"abc"` rechtsbündig auf einer Breite von 10 Zeichen dar und fülle links mit Sternen (`*`) auf:

  `*******abc`

- `"{0:+,}".format(1234567)`

  Stelle die ganze Zahl `123456789` mit positivem Vorzeichen dar und trenne jeweils drei Ziffern von rechts her durch ein Komma ab:

  `+1,234,567`

- `"{0:.4f}".format(123.456789)`

  Stelle die Gleitkommazahl `123.456789` auf vier Nachkommastellen gerundet dar.

  `123.4568`

- `"{0:.4e}".format(123.456789)`

  Stelle die Gleitkommazahl `123.456789` auf vier Nachkommastellen in der Exponentenschreibeise dar:

  `1.2346e+2`

- `"{0:0>8b}".format(15)`

  Stelle die ganze Zahl 15 rechtsbündig im Binärformat dar. Verwende 8 Positonen und fülle links mit Nullen auf:

  `00001111`