```
Formatierte Ausgabe in Python              (C) 2019-2021 T.Birnthaler OSTC GmbH
=============================

Doku:

  http://pyformat.info                                                      (Sehr hilfreiche B
eispielseite!)
  http://docs.python.org/3/library/functions.html#format                    format Function
  http://docs.python.org/3/library/stdtypes.html#old-string-formatting      printf-style Strin
g Formatting
  http://docs.python.org/3/library/stdtypes.html#printf-style-string-formatting   printf-style Strin
g Formatting
  http://docs.python.org/3/library/stdtypes.html#str.format                 String Methods
  http://docs.python.org/3/library/string.html#formatspec                   Format-specificati
on Mini-Language
  http://docs.python.org/3/library/string.html#formatstrings                Format String Synt
ax
  http://docs.python.org/3/library/string.html#string-formatting            Custom String Form
atting
  http://docs.python.org/3/library/string.html#string.Formatter             Formatter Class
  http://docs.python.org/3/reference/lexical_analysis.html#f-strings        Formatted String L
iterals
  http://docs.python.org/3/library/string.html#template-strings             Template Strings

In Python gibt es 3 Arten, FORMATIERTE Zeichenketten mit FESTEN Textteilen und
VARIABLEN Datenteilen darin zu erzeugen. Alle 3 beruhen auf dem Prinzip, in
eine SCHABLONE/ein TEMPLATE (FMT-String mit festem TEXT + PLATZHALTERN) Werte
im passenden Format einzubauen und den Ergebnisstring zurÃ¼ckzugeben:

 +------+-------------------+-----------------+---------------------+--------+
 | Stil | Modern Flex LÃ¤nge | Bezeichnung     | Syntax              | Platzh.|
 +------+-------------------+-----------------+---------------------+--------+
 | C    | ALT    NIED LANG  | Analog C-printf | "FMT" % (WERTE)     | %...   |
 | Py 1 | MODERN MITT LANG  | String-Funktion | "FMT".format(WERTE) | {...}  |
 | Py 2 | MODERN HOCH KOMP  | Format-String   | F"FMT+WERTE"        | {...}  |
 +------+-------------------+-----------------+---------------------+--------+
  (ALT/MODERN=veraltet/neu, NIED/MITT/HOCH=FlexibilitÃ¤t, LANG/KOMP=lang/kompakt)

Hier ein Beispiel fÃ¼r alle 3 Stile mit dem gleichen Ergebnis: Folgende 3 Werte

  z = 5678      # Ganzzahl
  k = 3.1415    # FlieÃM-^_kommazahl
  t = "hallo"   # String

werden per Platzhalter mit passendem Formattyp (d=decimal, f=float, s=string)
in einem String eingefÃ¼gt:

  print( "Zahl %6d, Wert %6.2f, Text %-10s" % (z, k, t))
  print( "Zahl {0:6d}, Wert {1:6.2f}, Text {2:10s}".format(z, k, t))
  print( "Zahl {a:6d}, Wert {b:6.2f}, Text {c:10s}".format(a=z, b=k, c=t))
  print(F"Zahl {z:6d}, Wert {k:6.2f}, Text {t:10s}")

Es ergibt sich jeweils der gleiche Ergebnisstring (Zahlen rechtsbÃ¼ndig, String
linksbÃ¼ndig):

  "Zahl   5678, Wert   3.14, Text hallo     "
        ------       ------       ----------
          6d          6.2f         -10s/10s

Die PLATZHALTER werden also per % oder {} dargestellt, der TYP-BUCHSTABE am
Ende eines Platzhalters legt den Datentyp des einzufÃ¼llenden Werts fest (Python
prÃ¼ft dies auch). Die WERTE zu den Platzhaltern werden festgelegt durch:

  1)  Reihenfolge im Tupel nach %-Zeichen (KEINE Umordnung + Mehrfachverwendung)
  2a) Position im .format()-Aufruf        (Umordnung + Mehrfachverw. mÃ¶glich)
  2b) Name im .format()-Aufruf            (Umordnung + Mehrfachverw. mÃ¶glich)
  3)  Variablenname im Platzhalter {}     (Umordnung + Mehrfachverw. mÃ¶glich)

Folgende TYP-BUCHSTABEN sind in allen Stilen verfÃ¼gbar:

  +-----+-------------+-------------------------------------------------------+
  | Typ | Name        | Bedeutung                                             |
  +-----+-------------+-------------------------------------------------------+
  | s   | string      | Zeichenkette (DEFAULT, weglassbar)                    |
  | c   | character   | Zeichen per Code-Nummer (Unicode)                     |
  +-----+-------------+-------------------------------------------------------+
  | d   | decimal     | Ganzzahl im Dezimalformat                             |
  | i   | integer     | Ganzzahl im Dezimalformat (nur im C-Stil, analog "d") |
  | n   | number      | Ganzzahl analog "d" (aber Trennzeichen gemÃ¤ÃM-^_ locale)   |
  +-----+-------------+-------------------------------------------------------+
  | b   | binary      | Ganzzahl im BinÃ¤rformat (nur Ziffern 0/1)             |
  | o   | octal       | Ganzzahl im Oktalformat (nur Ziffern 0-7)             |
```

```
  | x X | hexadecimal | Ganzzahl im Hexadezimalformat (x=abcdef, X=ABCDEF)     |
  +-----+-------------+-------------------------------------------------------+
  | f F | float       | FlieÃM-^_kommazahl im Festpunktformat (inf/nan <-> INF/NAN)|
  | e E | exponent    | FlieÃM-^_kommazahl im Exponentialformat mit e/E-Zeichen    |
  | g G | general     | "f/F" oder "e/E" je nachdem wo mehr Ziffern dargestellt|
  | n   | number      | FlieÃM-^_kommazahl analog "g" (Trennzeichen gemÃ¤ÃM-^_ locale)  |
  +-----+-------------+-------------------------------------------------------+
  | %   | percent     | FlieÃM-^_kommazahl analog "f" * 100 + %-Zeichen danach     |
  +-----+-------------+-------------------------------------------------------+

Die Typinformation kann um weitere Angabe ergÃ¤nzt werden, mit denen minimale +
maximale Breite, FÃ¼llzeichen, Ausrichtung, Anzahl Nachkommastellen und andere
Eigenschaften der Ausgabe kontrolliert werden.


C-printf-Stil
-------------
Formatelemente beginnen mit dem Zeichen "%" und enden mit einem Typbuchstaben
(ohne Zwischenraum, ein Prozentzeichen ist zu verdoppeln):

  % (MAP) FLAGS WIDTH .PREC LENMOD TYPE

Bedeutung:

  +-------+-------------------------------------------------------------------+
  | MAP   | Dictionary Key (Wert muss ein Dictionary sein)                    |
  | FLAGS | "#" Alternative Form ("0o", "0x", "0X")                           |
  |       | "0" Mit "0" statt Leerzeichen auffÃ¼llen                           |
  |       | "-" LinksbÃ¼ndig (statt rechtsbÃ¼ndig)                             |
  |       | " " Vorzeichen "+" weglassen                                      |
  |       | "+" Vorzeichen "+" ausgeben                                       |
  | WIDTH | Minimale Breite (mit " " oder "0" aufgefÃ¼llt)                    |
  | .PREC | Maximale Breite bei Strings / Anzahl Nachkommastellen bei Zahlen  |
  | LENMOD| "hlL" (in Python irrelevant)                                      |
  | TYPE  | "scdinboxXfFeEgG%" analog obiger Typtabelle                       |
  +-------+-------------------------------------------------------------------+

Beispiele:

    zahl = 1234567.89
    "%+12.1f"  % zahl   # --> '  +1234567.9'
    "%+012.1f" % zahl   # --> '+001234567.9'
    "%-+12.1f" % zahl   # --> '+1234567.9  '
    "%--12.1f" % zahl   # --> '1234567.9   '
    "%- 12.1f" % zahl   # --> ' 1234567.9  '
    "%-12.1f"  % zahl   # --> '1234567.9   '
    "%-012.1f" % zahl   # --> '1234567.9   '
    "%012.1f"  % zahl   # --> '0001234567.9'
    "%f"       % zahl   # --> '1234567.890000'

    text = "hallowelt"
    "%s"       % text   # --> 'hallowelt'
    "%12s"     % text   # --> '   hallowelt'
    "%-12s"    % text   # --> 'hallowelt   '
    "%012s"    % text   # --> '   hallowelt'
    "%12.12s"  % text   # --> '   hallowelt'
    "%6.6s"    % text   # --> 'hallow'


Python 1 Stil
-------------
Die Formatangabe hat folgende Form (ohne Zwischenraum, INDEX ist eine
Positionsnummer (startet mit 0) oder ein Name, dessen Wert eingefÃ¼llt wird,
ein "{" oder "}" ist zu verdoppeln, !CONV und :SPEC sind optional):

  "{}"
  "{" INDEX/NAME ":"}"
  "{" INDEX/NAME ":" SPEC "}"
  "{" INDEX/NAME "!" CONV ":" SPEC "}"

Im Python-Stil 1 steht im Platzhalter vor einem ":" ein INDEX (startet mit 0),
welche die POSITION in der Werte-Liste im .format(WERTE)-Aufruf angibt
(Reihenfolge der Nummern beliebig, gleiche Nummer mehrfach erlaubt). Ohne Index
werden die Werte von links nach rechts in die Platzhalter eingefÃ¼llt (":" ist
trotzdem notwendig!). Alternativ sind auch Namen fÃ¼r die format()-Parameter
vergebbar, die vor dem ":" als Referenz verwendbar sind:

  F"Zahl {0:6d}, Wert {1:6.2f}, Text {2:10s}".format(z, k, t)       # Index expl.
  F"Zahl {:6d}, Wert {:6.2f}, Text {:10s}".format(z, k, t)          # Index autom.
  F"Zahl {a:6d}, Wert {b:6.2f}, Text {c:10s}".format(a=z, b=k, c=t) # Name

Ergibt:
```

```
  "Zahl   5678, Wert   3.14, Text hallo     "
  "Zahl   5678, Wert   3.14, Text hallo     "
  "Zahl   5678, Wert   3.14, Text hallo     "
        ------       ------       ----------
          6d          6.2f           10s
```

Im einfachsten Fall genÃ¼gt "{}" als Platzhalter, diese werden von links nach rechts mit den Werten in .format(...) in der notwendigen Breite gefÃ¼llt (eine TypprÃ¼fung findet hier nicht statt):

```
  F"Zahl {}, Wert {}, Text {}".format(z, k, t)
```

Ergibt:

```
  "Zahl 5678, Wert 3.1415, Text hallo"
        ----       ------       -----
```

Python 2 Stil

Die Formatangabe hat folgende Form (ohne Zwischenraum, INDEX ist eine Positionsnummer (startet mit 0) oder ein Name, dessen Wert eingefÃ¼llt wird, ein "{" oder "}" ist zu verdoppeln, !CONV und :SPEC sind optional):

```
  "{" EXPR ":"}"
  "{" EXPR ":" SPEC "}"
  "{" EXPR "!" CONV ":" SPEC "}"
```

Bedeutung:

```
  +------+-------------------------------------------------------------+
  | EXPR | Zu formatiertendes Objekt (Variablenname/Rechenausdruck)    |
  | CONV | Festlegung der Konvertierung gemÃ¤ÃM-^_ folgender Tabelle     |
  | SPEC | Festlegung der Formatierung gemÃ¤ÃM-^_ folgender Tabelle      |
  +------+-------------------------------------------------------------+
```

CONF konvertiert den Wert gemÃ¤ÃM-^_ folgender Regeln:

```
  +----+---------+-----------------------------------+
  | !a | ascii   | ascii() auf Wert anwenden         |
  | !r | repr    | repr() auf Wert anwenden          |
  | !s | string  | str() auf Wert anwenden (Default) |
  +----+---------+-----------------------------------+
```

SPEC legt die Formatierung mit folgenden Komponenten fest (in dieser Reihenfolge, ohne Zwischenraum, alles optional auÃM-^_er FILL erfordert ALIGN):

```
  +-------+-------------------------------------------------------------------+
  | FILL  | FÃ¼llzeichen (Standard Leerzeichen " ", "}" nicht erlaubt          |
  | ALIGN | "<" (left), ">" (right), "^" (center), "=" (zw. Vorz. und Zahl)   |
  | SIGN  | "+" ("+"/"-" erzwingen), "-" ("-" falls nÃ¶tig), " " (" "/"-")     |
  | #     | Setzt "0b", "0o" oder "0x" vor Ganzzahlen (abhÃ¤ngig von TYPE)      |
  | 0     | Zahlen mit "0" auffÃ¼llen (FILL ist allgemeiner)                    |
  | WIDTH | Minimale Breite (Mit " ", "0" oder FILL aufgefÃ¼llt)               |
  | ,     | Tausendertrenner erzeugen (3er-Gruppen)                            |
  | .PREC | Maximale Breite bei Strings / Anzahl Nachkommastellen bei Zahlen   |
  | TYPE  | "scdinboxXfFeEgG%" analog obiger Typtabelle                        |
  +-------+-------------------------------------------------------------------+
```

Beispiel:

```
  F"Zahl {z:6d}, Wert {k:6.2f}, Text {t:10s}")               # Variable
  F"Zahl {z*5+2:6d}, Wert {k/5.0:6.2f}, Text {t+'X':10s}")   # Rechenausdruck
```

Ergibt:

```
  "Zahl   5678, Wert   3.14, Text hallo     "
  "Zahl  28392, Wert   0.63, Text halloX    "
        ------       ------       ----------
          6d          6.2f           10s
```

Im Python Stil 2 lassen sich mit weiteren (eingeschachtelten) Klammern {...} beliebige Teile der Format-Angabe per Variable oder Rechenausdruck fÃ¼llen (Makro-Logik):

```
  breite = 10
  nkst   = 3
  typ    = "d"
  F"Zahl {z:{breite}{typ}}, Wert {k:{breite}.{nkst}f}, Text {t:{breite}s}"
```

Ergibt:

```
  "Zahl       5678, Wert      3.142, Text hallo     "
        ----------        ----------        ----------
           10d               10.3f             10s
```

Weitere Beispiele zum Python Stil 1 + 2:

```
    zahl = 1234567.89

    "{:*=+12.1f}".format(zahl) # --> '+**1234567.9'
    "{:*> 12.1f}".format(zahl) # --> '** 1234567.9'
    "{:-012.1f}".format(zahl)  # --> '0001234567.9'
    "{:,.2f}".format(zahl)     # --> '1,234,567.89'

    F"{zahl:*=+12.1f}"         # --> '+**1234567.9'
    F"{zahl:*>+12.1f}"         # --> '** 1234567.9'
    F"{zahl:-012.1f}"          # --> '0001234567.9'
    F"{zahl:,.2f}"             # --> '1,234,567.89'
```