Built-in Functions von Python (C) 2015-2021 T.Birnthaler OSTC GmbH
==========

In Python sind 69 Funktionen fest eingebaut ("built-in"), sie kÃ¶nnen Ã¼berschrieben werden, sollten dies aber nicht unbedingt (da sonst z.B. die Funktion "print()" oder Datentypen wie "list" nicht mehr funktionieren).

Doku --> http://docs.python.org/3/library/functions.html
http://docs.python.org/3/library/builtins.html

Diese GENERISCHEN FUNKTIONEN sind auf alle oder zumindest mehrere Datentypen anwendbar, daher sind sie nicht als Methode eines Datentyps realisiert und nicht in Module ausgelagert.

Ihnen ist gemeinsam, dass Sie KEINE ÃM-^DNDERUNGEN an den beim Aufruf Ã¼bergebenen Parametern durchfÃ¼hren, sondern ein Ergebnis zurÃ¼ckgeben (reine/pure Functions, einzige Ausname: delattr() und setattr()).

Eine Liste dieser Funktionen erhÃ¤lt man folgendermaÃM-^_en:

```
pydoc builtins        # Auf der Kommandozeile
import builtins       # Im Python-Skript
dir(builtins)         # Im Python-Skript
```

Zugriff auf die in Python direkt eingebauten Objekte (Funktionen, Konstanten, ...) ist Ã¼ber das Modul "builtins" grundsÃ¤tzlich immer mÃ¶glich:

```
import builtins
builtins.open(...)    # Statt open(...)
```

Funktion	Beschreibung
id(OBJ)	IdentitÃ¤t von Objekt (Ganzzahl = Speicheradresse)
type(OBJ)	Datentyp von Objekt
dir(OBJ)	Elemente von (lokalem) Namensraum/Objektattribut
hash(OBJ)	Hashwert zu Objekts erstellen --> OBJ.__hash__()
help(OBJ)	Doku zu Objekt ausgeben --> OBJ.__doc__
abs(N)	Absolutwert von Zahl N
divmod(A,B)	Ergibt Quotient/Rest-Tupel (A // B, A % B)
round(N,M)	Zahl N runden auf M Nachkommastellen (oder Ganzzahl)
pow(X,Y,Z)	X hoch Y (modulo Z)
all(ITER)	True falls alle Elemente True (oder ITER leer)
any(ITER)	False falls alle Elemente False (oder ITER leer)
max(ITER)	GrÃ¶Ã-^_tes Element (Vergleich < muss funktionieren)
min(ITER)	Kleinstes Element (Vergleich < muss funktionieren)
sum(ITER,START)	Summe der Elemente (mÃ¼ssen Zahlen sein)
repr(OBJ)	Datentyp-Konvertierung nach "str" (maschinenlesbar)
ascii(OBJ)	repr() + non-ASCII Zeichen escaped (\x \u \U)
chr(N)	Zeichen mit Unicode N
ord(STR)	Unicode des 1. Zeichens von STR
bin(N)	Zahl N als BinÃ¤r-String "0b..." zurÃ¼ckgeben
hex(N)	Zahl N als Hexadezimal-String "0x..." zurÃ¼ckgeben
oct(N)	Zahl N als Oktal-String "0o..." zurÃ¼ckgeben
bool(VAL)	Datentyp-Konvertierung nach "bool"
bytearray(SRC,ENC,ERR)	Datentyp-Konvert. nach "bytearray" (mutable int seq)
bytes(SRC,ENC,ERR)	Datentyp-Konvert. nach "bytes" (immutable int seq)
complex(REAL,IMAG=0)	Datentyp-Konvertierung nach "complex"
dict(OBJ)	Datentyp-Konvertierung nach "dict"
float(VAL)	Datentyp-Konvertierung nach "float"
frozenset(ITER)	Datentyp-Konvertierung nach "frozenset"
int(OBJ,BASE=10)	Datentyp-Konvertierung nach "int" (zur Basis BASE)
list(ITER)	Datentyp-Konvertierung nach "list"
memoryview(OBJ)	Datentyp-Konvertierung nach "memoryview"
set(ITER)	Datentyp-Konvertierung nach "set"
str(OBJ)	Datentyp-Konvertierung nach "str"
tuple(ITER)	Datentyp-Konvertierung nach "tuple"
globals()	Globale Variablen im aktuellen Namensraum (Modul)
locals()	Lokale Variablen im aktuellen Namensraum (Funktion)
vars() vars(OBJ)	Analog locals() bzw. OBJ.__dict__
delattr(OBJ,ATTR)	Attribut zu Objekt lÃ¶schen --> del OBJ.ATTR
getattr(OBJ,ATTR,DFLT)	Wert zu Attribut von Objekt holen (DFLT falls fehlt)
hasattr(OBJ,ATTR)	True falls Objekt Attribut hat, sonst False
setattr(OBJ,ATTR,VAL)	Attribut in Objekt auf Wert setzen

(C) 2024 OSTC GmbH (http://www.ostc.de)

callable(OBJ)	True falls Objekt aufrufbar (Func/Class/Object)
isinstance(OBJ,CLS)	Objekt ist Instanz von Kl. oder einer ihrer Unterkl.
issubclass(CLS,CLS2)	Klasse ist Unterklasse (auch von sich selbst)
super(TYPE,OBJ/TYPE)	Elternklasse oder Geschwisterklasse
object()	Neues leeres Objekt (Basisklasse aller Klassen)
@property(GET,SET,DEL,DOC)	Property-Attribut erzeugen (verpackt 4 Methoden)
@classmethod()	Decorator fÃ¼r Klassenmethode C.f(cls)
@staticmethod()	Decorator fÃ¼r Statische Methode C.f()
open(FILE,MODE="r",BUF=-1,ENC=None,ERR=None,NL=None,CLOSEFD=True,OPENER=None)	Datei Ã¶ffnen und Fileobjekt zurÃ¼ckgeben
input(PROMPT)	Terminal-Eingabe als "str" einlesen
print(LIST,SEP=" ",END="\n",FILE=sys.stdout,FLUSH=False)	Alle Elemente von LIST per str() ausgeben
format(VAL,FORMAT)	Formatierter Wert
len(SEQ)	Anzahl Elemente (Sequenz, Range, Collection)
reversed(SEQ)	Umgekehrten Iterator zurÃ¼ckgeben
sorted(ITER,KEY,REV)	Sortierte Folge zurÃ¼ckgeben (Vgl. < muss funkt.)
iter(OBJ,SENTINEL)	Iterator-Objekt zurÃ¼ckgeben --> __iter__()
next(ITER,DEFAULT)	NÃ¤chstes Element aus Iterator holen --> __next__()
range(BEG=0,END,STEP=1)	Zahlenliste BEG...END-1 erzeugen
slice(BEG=0,END,STEP=1)	Slice-Objekt SEQ[BEG:END:STEP] erzeugen
enumerate(ITER,BEG=0)	Elemente mit aufsteigender Nummer versehen (0/BEG)
filter(FUNC,ITER)	Elemente filtern gemÃ¤ÃM-^_ Funktion --> True
map(FUNC,ITER)	Elemente abbilden gemÃ¤ÃM-^_ Funktion
zip(*ITER)	2 oder mehr Folgen zu Folge von Tupeln verschrÃ¤nken
breakpoint(*, **)	Sprung in Debugger sys.breakpointhook() --> pdb.set_trace()
compile(SRC,FILE,MOD,FLG,DONTINHERIT=False,OPT=-1)	Kompil. SRC in Code oder AST Objekt --> exec/eval()
eval(STR,GLOBAL,LOCAL)	String/Code-Objekt als Python-Ausdruck auswerten
exec(OBJ,GLOBAL,LOCAL)	String/Code-Objekt als Python-Skript ausfÃ¼hren
__import__(NAME,...)	Von "import" aufgerufen

HINWEIS: ITER steht fÃ¼r ein ITERABLE, also ein Collection-Objekt, dessen
Elemente per Iterator-Protokoll (iter(), next(), StopIteration) durchlaufen
werden kÃ¶nnen (z.B. Sequenz, Comprehension, Generator, Range, File, ...).

```
  for ELEM in ITER:
      print(ELEM)
```