

Python "Magic Methods"

(C) 2018-2021 T.Birnthaler OSTC GmbH

Python kennt 94 magische Methoden ("dunder methods" = double underscore), die Python automatisch aufruft, wenn die entsprechende Operation mit einem Objekt (oder einer Klasse) durchgeführt wird (P2/P3 = nur in Python 2/3).

Doku --> http://docs.python.org/3/reference/datamodel.html#object.__new__

Magische Methode	Aufruf durch ...	Bedeutung
<code>__new__</code> <code>__init__</code> <code>__del__</code> <code>__slots__</code>	CLS() CLS() del OBJ garbage collector CLS attributes	Subclass immutable type Konstruktor Destruktor Attribute
<code>__enter__</code> <code>__exit__</code>	with ... as ... with ... as ...	Kontext-Manager "
<code>__iter__</code> <code>__next__</code>	for ... in ..., enumerate() for ... in ..., enumerate()	Iterator-Protokoll "
<code>__getinitargs__</code> <code>__getnewargs__</code> <code>__getstate__</code> <code>__setstate__</code> <code>__reduce__</code> <code>__reduce_ex__</code>	pickle.dump()/restore() pickle.dump()/restore() pickle.dump()/restore() pickle.dump()/restore() pickle.dump()/restore() pickle.dump()/restore()	Pickling-Protokoll " " " " "
<code>__copy__</code> <code>__deepcopy__</code>	copy.copy() copy.deepcopy()	Flache Kopie (shallow) Tiefe Kopie (deep)
<code>__repr__</code> <code>__str__</code> <code>__format__</code> <code>__unicode__</code> P2 <code>__hash__</code> <code>__nonzero__</code> P2	repr() str() F"... "..."format() unicode() hash() bool() on instances	Maschinen-Repräsentation Menschliche Darstellung Formatierung in P3 <code>__str__</code> + <code>__byte__</code> in P3 renamed <code>__bool__</code>
<code>__bool__</code> <code>__bytes__</code> <code>__complex__</code> <code>__int__</code> <code>__long__</code> P2 <code>__float__</code>	bool() bytes() complex() int() long() float()	Typ-Konvertierung " " " " "
<code>__oct__</code> <code>__hex__</code> <code>__bin__</code>	oct() hex() bin()	Zahlen-Konvertierung " "
<code>__index__</code> <code>__dir__</code> <code>__call__</code> <code>__coerce__</code> P2	OBJ[N:M:S] dir() OBJ() called as function	Slice-Operator Objekt-Attributliste x() --> x.__call__() Mixed mode arithmetic
<code>__getattr__</code> <code>__getattribute__</code> <code>__setattr__</code> <code>__delattr__</code>	OBJ.ATTR getattr() OBJ.ATTR getattr() OBJ.ATTR = VAL setattr() del OBJ.ATTR delattr()	Objekt-Attribute " " "
<code>__get__</code> <code>__set__</code> <code>__delete__</code> <code>__set_name__</code>		Deskriptor-Klasse TODO " " "
<code>__init_subclass__</code> <code>__instancecheck__</code> <code>__subclasscheck__</code> <code>__class_getitem__</code>	isinstance() issubclass()	TODO TODO
<code>__len__</code> <code>__length_hint__</code> <code>__reversed__</code> <code>__contains__</code>	len() reversed() in not in	Anzahl Container-Elemente TODO Reihenfolge umdrehen Enthält (nicht)
<code>__getitem__</code> <code>__missing__</code> <code>__setitem__</code> <code>__delitem__</code>	OBJ[KEY] OBJ[KEY] OBJ[KEY] = VALUE del OBJ[KEY]	Indezzugriff auf Element dict-Subklasse " "
<code>__eq__</code>	x == y	Werte-Vergleich

<code>__ne__</code>	<code>x != y</code>	"
<code>__lt__</code>	<code>x < y</code>	"
<code>__le__</code>	<code>x <= y</code>	"
<code>__gt__</code>	<code>x > y</code>	"
<code>__ge__</code>	<code>x >= y</code>	"
<code>__cmp__</code> P2	<code>x </==/> y --> -1/0/1</code>	"
<code>__add__</code>	<code>+</code>	Arithmetische Operatoren
<code>__sub__</code>	<code>-</code>	"
<code>__mul__</code>	<code>*</code>	"
<code>__matmul__</code>	<code>@</code>	"
<code>__div__</code> P2	<code>/</code>	"
<code>__truediv__</code> P3	<code>/</code>	"
<code>__floordiv__</code> P3	<code>//</code>	"
<code>__mod__</code>	<code>%</code>	"
<code>__divmod__</code>	<code>divmod()</code>	"
<code>__pow__</code>	<code>** pow()</code>	"
<code>__lshift__</code>	<code><<</code>	"
<code>__rshift__</code>	<code>>></code>	"
<code>__and__</code>	<code>&</code>	"
<code>__or__</code>	<code> </code>	"
<code>__xor__</code>	<code>^</code>	"
<code>__radd__</code>	<code>+</code>	Reflektierte arith. Op.
<code>__rsub__</code>	<code>-</code>	"
<code>__rmul__</code>	<code>*</code>	"
<code>__rmatmul__</code>	<code>@</code>	"
<code>__rdiv__</code> P2	<code>/</code>	"
<code>__rtruediv__</code> P3	<code>/</code>	"
<code>__rfloordiv__</code> P3	<code>//</code>	"
<code>__rmod__</code>	<code>%</code>	"
<code>__rdivmod__</code>	<code>divmod()</code>	"
<code>__rpow__</code>	<code>**</code>	"
<code>__rlshift__</code>	<code><<</code>	"
<code>__rrshift__</code>	<code>>></code>	"
<code>__rand__</code>	<code>&</code>	"
<code>__ror__</code>	<code> </code>	"
<code>__rxor__</code>	<code>^</code>	"
<code>__iadd__</code>	<code>+=</code>	Erweiterte Zuweisung
<code>__isub__</code>	<code>-=</code>	"
<code>__imul__</code>	<code>*=</code>	"
<code>__imatmul__</code>	<code>@=</code>	"
<code>__idiv__</code> P2	<code>/=</code>	"
<code>__itruediv__</code> P3	<code>/=</code>	"
<code>__ifloordiv__</code> P3	<code>//=</code>	"
<code>__imod__</code>	<code>%=</code>	"
<code>__ipow__</code>	<code>**=</code>	"
<code>__ilshift__</code>	<code><<=</code>	"
<code>__irshift__</code>	<code>>>=</code>	"
<code>__iand__</code>	<code>&=</code>	"
<code>__ior__</code>	<code> =</code>	"
<code>__ixor__</code>	<code>^=</code>	"
<code>__neg__</code>	<code>-</code>	Unäre arithmetische Op.
<code>__pos__</code>	<code>+</code>	"
<code>__abs__</code>	<code>abs()</code>	"
<code>__invert__</code>	<code>~</code>	"
<code>__round__</code>	<code>round()</code>	Fließkomma-Konvertierung
<code>__floor__</code>	<code>floor()</code>	"
<code>__ceil__</code>	<code>ceil()</code>	"
<code>__trunc__</code>	<code>trunc()</code>	"

HINWEISE:

- * Die Zuweisung "=" ist KEIN Operator und kann daher nicht überladen werden (gibt immer ID zu einem Objekt zurück).
- * "is" und "is not" vergleichen die Objekt-Identität per ID und kann auch nicht überladen werden. Abkürzung für:


```
o1 is o2      # id(o1) == id(o2)
o1 is not o2  # id(o1) != id(o2)
```
- * `__iOP__` = Intrinsic Operator, aufgerufen bei verkürzter Zuweisung `a OP= b`
- * `__rOP__` = Reflected Operator, aufgerufen falls die von `o1 OP o2` aufgerufene Funktion `type(o1).__OP__(o2)` den Wert "NotImplemented" zurückgibt oder falls die Funktion `type(o1).__OP__()` nicht vorhanden ist.