

ds

For ex-
nation for
a syntax

the author
overload-
icular, the
which takes
the expres-
ect a of the
r operators.

types, as in
and. In this
nt definition
d. However,
class defini-
ed `__rmul__`
ass to support
that the exist-
is new class).
to define dif-
n operation is

pecially named
hen applied to
lly a call to the
stance of a user-
ave known how
specially named
esentation.
t an int, float, or
sion to a Boolean
e used even when
ser-defined class

Common Syntax	Special Method Form
<code>a + b</code>	<code>a.__add__(b);</code> alternatively <code>b.__radd__(a)</code>
<code>a - b</code>	<code>a.__sub__(b);</code> alternatively <code>b.__rsub__(a)</code>
<code>a * b</code>	<code>a.__mul__(b);</code> alternatively <code>b.__rmul__(a)</code>
<code>a / b</code>	<code>a.__truediv__(b);</code> alternatively <code>b.__rtruediv__(a)</code>
<code>a // b</code>	<code>a.__floordiv__(b);</code> alternatively <code>b.__rfloordiv__(a)</code>
<code>a % b</code>	<code>a.__mod__(b);</code> alternatively <code>b.__rmod__(a)</code>
<code>a ** b</code>	<code>a.__pow__(b);</code> alternatively <code>b.__rpow__(a)</code>
<code>a << b</code>	<code>a.__lshift__(b);</code> alternatively <code>b.__rlshift__(a)</code>
<code>a >> b</code>	<code>a.__rshift__(b);</code> alternatively <code>b.__rrshift__(a)</code>
<code>a & b</code>	<code>a.__and__(b);</code> alternatively <code>b.__rand__(a)</code>
<code>a ^ b</code>	<code>a.__xor__(b);</code> alternatively <code>b.__rxor__(a)</code>
<code>a b</code>	<code>a.__or__(b);</code> alternatively <code>b.__ror__(a)</code>
<code>a += b</code>	<code>a.__iadd__(b)</code>
<code>a -= b</code>	<code>a.__isub__(b)</code>
<code>a *= b</code>	<code>a.__imul__(b)</code>
...	...
<code>+a</code>	<code>a.__pos__()</code>
<code>-a</code>	<code>a.__neg__()</code>
<code>~a</code>	<code>a.__invert__()</code>
<code>abs(a)</code>	<code>a.__abs__()</code>
<code>a < b</code>	<code>a.__lt__(b)</code>
<code>a <= b</code>	<code>a.__le__(b)</code>
<code>a > b</code>	<code>a.__gt__(b)</code>
<code>a >= b</code>	<code>a.__ge__(b)</code>
<code>a == b</code>	<code>a.__eq__(b)</code>
<code>a != b</code>	<code>a.__ne__(b)</code>
<code>v in a</code>	<code>a.__contains__(v)</code>
<code>a[k]</code>	<code>a.__getitem__(k)</code>
<code>a[k] = v</code>	<code>a.__setitem__(k,v)</code>
<code>del a[k]</code>	<code>a.__delitem__(k)</code>
<code>a(arg1, arg2, ...)</code>	<code>a.__call__(arg1, arg2, ...)</code>
<code>len(a)</code>	<code>a.__len__()</code>
<code>hash(a)</code>	<code>a.__hash__()</code>
<code>iter(a)</code>	<code>a.__iter__()</code>
<code>next(a)</code>	<code>a.__next__()</code>
<code>bool(a)</code>	<code>a.__bool__()</code>
<code>float(a)</code>	<code>a.__float__()</code>
<code>int(a)</code>	<code>a.__int__()</code>
<code>repr(a)</code>	<code>a.__repr__()</code>
<code>reversed(a)</code>	<code>a.__reversed__()</code>
<code>str(a)</code>	<code>a.__str__()</code>

Table 2.1: Overloaded operations, implemented with Python's special methods.