* The twist in this puzzle is to understand the powerful zip function. The zip function usually generates a sequence of tuples from two given sequence arguments - pairing the i-th element of the first with the i-th element of the second sequence. However, when used with the \* operator, this procedure is reversed to generate the two sequences from a given list of tuples. Hence, it becomes an unzip function. We store the two sequences in the variables a and b and concatenate the characters in the sequence using the join method on the empty string, i.e., we glue together the sequence elements without inserting any string in-between.
* If you don’t want characters prefaced by \ to be interpreted as special characters, you can use raw strings by adding an r before the first quote: print(r'C:\some\name')
* The Python interpreter maintains a symbol table that stores all function definitions, i.e., mappings from function names to function objects. In this way, the interpreter can relate each occurrence of the function name to the defined function object. **Just remember: a single function object can have zero, one, or even many names.**
* Unlike strings, which are immutable, lists are a mutable type, i.e. it is possible to change their content.
* **Append** will only accept one argument, therefore, when we try to add a list of three items, as we do above, append adds them as a list item. [**1, 2, 3, [4, 5, 6]].**

**Extend**, on the other hand, can accept more than one argument, therefore, it adds the three items in the second list separately. **[1, 2, 3, 4, 5, 6]**

* The **list.clear()** method removes all elements from an existing list. The list becomes empty again.
* The **range(x)** creates a range starting at 0 and ending at x-1. **E.G. range(4) creates range as 0, 1, 2, 3.**
* The function swap only returns its two arguments in reversed order. Since we assign the return values of **swap()** to a and b, **a is set to the value of b and b to the value of a**.

| **Operation** | **Result** | **Notes** | **Full documentation** |
| --- | --- | --- | --- |
| x + y | sum of *x* and *y* |  |  |
| x - y | difference of *x* and *y* |  |  |
| x \* y | product of *x* and *y* |  |  |
| x / y | quotient of *x* and *y* |  |  |
| x // y | floored quotient of *x* and *y* | (1) |  |
| x % y | remainder of x / y | (2) |  |
| -x | *x* negated |  |  |
| +x | *x* unchanged |  |  |
| abs(x) | absolute value or magnitude of *x* |  | [abs()](https://docs.python.org/3/library/functions.html#abs) |
| int(x) | *x* converted to integer | (3)(6) | [int()](https://docs.python.org/3/library/functions.html#int) |
| float(x) | *x* converted to floating point | (4)(6) | [float()](https://docs.python.org/3/library/functions.html#float) |
| complex(re, im) | a complex number with real part *re*, imaginary part *im*. *im* defaults to zero. | (6) | [complex()](https://docs.python.org/3/library/functions.html#complex) |
| c.conjugate() | conjugate of the complex number *c* |  |  |
| divmod(x, y) | the pair (x // y, x % y) | (2) | [divmod()](https://docs.python.org/3/library/functions.html#divmod) |
| pow(x, y) | *x* to the power *y* | (5) | [pow()](https://docs.python.org/3/library/functions.html#pow) |
| x \*\* y | *x* to the power *y* | (5) |  |

Notes:

1. Also referred to as integer division. The resultant value is a whole integer, though the result’s type is not necessarily int. The result is always rounded towards minus infinity: 1//2 is 0, (-1)//2 is -1, 1//(-2) is -1, and (-1)//(-2) is 0.
2. Not for complex numbers. Instead convert to floats using [abs()](https://docs.python.org/3/library/functions.html#abs) if appropriate.
3. Conversion from floating point to integer may round or truncate as in C; see functions [math.floor()](https://docs.python.org/3/library/math.html#math.floor) and [math.ceil()](https://docs.python.org/3/library/math.html#math.ceil) for well-defined conversions.
4. float also accepts the strings “nan” and “inf” with an optional prefix “+” or “-” for Not a Number (NaN) and positive or negative infinity.
5. Python defines pow(0, 0) and 0 \*\* 0 to be 1, as is common for programming languages.
6. The numeric literals accepted include the digits 0 to 9 or any Unicode equivalent (code points with the Nd property).