

6.2.6 Set displays

A set display is denoted by curly braces and distinguishable from dictionary displays by the lack of colons separating keys and values:

```
set_display ::= "{" (starred_list | comprehension) "}"
```

A set display yields a new mutable set object, the contents being specified by either a sequence of expressions or a comprehension. When a comma-separated list of expressions is supplied, its elements are evaluated from left to right and added to the set object. When a comprehension is supplied, the set is constructed from the elements resulting from the comprehension.

An empty set cannot be constructed with `{ }`; this literal constructs an empty dictionary.

6.2.7 Dictionary displays

A dictionary display is a possibly empty series of key/datum pairs enclosed in curly braces:

```
dict_display      ::= "{" [key_datum_list | dict_comprehension] "}"
key_datum_list    ::= key_datum ("," key_datum)* [","]
key_datum         ::= expression ":" expression | "*" or_expr
dict_comprehension ::= expression ":" expression comp_for
```

A dictionary display yields a new dictionary object.

If a comma-separated sequence of key/datum pairs is given, they are evaluated from left to right to define the entries of the dictionary: each key object is used as a key into the dictionary to store the corresponding datum. This means that you can specify the same key multiple times in the key/datum list, and the final dictionary's value for that key will be the last one given.

A double asterisk `**` denotes *dictionary unpacking*. Its operand must be a *mapping*. Each mapping item is added to the new dictionary. Later values replace values already set by earlier key/datum pairs and earlier dictionary unpackings.

New in version 3.5: Unpacking into dictionary displays, originally proposed by [PEP 448](#).

A dict comprehension, in contrast to list and set comprehensions, needs two expressions separated with a colon followed by the usual “for” and “if” clauses. When the comprehension is run, the resulting key and value elements are inserted in the new dictionary in the order they are produced.

Restrictions on the types of the key values are listed earlier in section [The standard type hierarchy](#). (To summarize, the key type should be *hashable*, which excludes all mutable objects.) Clashes between duplicate keys are not detected; the last datum (textually rightmost in the display) stored for a given key value prevails.

Changed in version 3.8: Prior to Python 3.8, in dict comprehensions, the evaluation order of key and value was not well-defined. In CPython, the value was evaluated before the key. Starting with 3.8, the key is evaluated before the value, as proposed by [PEP 572](#).

6.2.8 Generator expressions

A generator expression is a compact generator notation in parentheses:

```
generator_expression ::= "(" expression comp_for ")"
```

A generator expression yields a new generator object. Its syntax is the same as for comprehensions, except that it is enclosed in parentheses instead of brackets or curly braces.

Variables used in the generator expression are evaluated lazily when the `__next__()` method is called for the