Expressions and statements

Software Development & Python

Nick Levashov, 2021

girafe ai

Expressions

```
>>> 10 / 5.0
>>> 2 ** 10

>>> a[:-1]

>>> foo(bar)

>>> a <= b

>>> a or b and c
```

Statements

```
>>> a = 10

>>> return a

>>> break
```

Statements

```
>>> a = 10

>>> return a

>>> break


>>> if a is b:
>>>     print('a is b')

>>> for i in range(10):
>>>     print(f'step #{i}')

>>> while True:
>>>     print('Can\'t stop')

>>> def foo(bar):
>>>     print('go to', bar)
```

Expressions

girafe
ai

Atoms

Literals

- string, bytes, integer, floating point number, complex number

Identifiers

- variable names, keywords

Arithmetic

```
>>> x + y
>>> x - y
>>> x * y
>>> x / y
>>> x % y
>>> x // y
>>> x ** y
```

Arithmetic

```
>>> x + y
>>> x - y
>>> x * y
>>> x / y
>>> x % y
>>> x // y
>>> x ** y
```

```
>>> x & y
>>> x | y
>>> +x
>>> -x
>>> ~x
>>> x ^ y
>>> x >> y
>>> x << y
```

Call

```
>>> func(arg1, arg2)
```

Call

```
>>> func(arg1, arg2)

expression (
    arg1, arg2, ...,
    kwarg1=kwarg1, kwarg2=kwarg2, ...
)
```

Call

```
>>> func(arg1, arg2)

expression (
    arg1, arg2, ...,
    kwarg1=kwarg1, kwarg2=kwarg2, ...
)

>>> func = 'ABC'.lower
>>> func()
'abc'
```

Call

```
>>> func(arg1, arg2)

expression (
    arg1, arg2, ...,
    kwarg1=kwarg1, kwarg2=kwarg2, ...
)

>>> func = 'ABC'.lower
>>> func()
'abc'

>>> (get_func())(arg)
```

girafe
ai

Displays

```
>>> [1, 2, 3, 'abc']
>>> {'Jack', 'Mike', 'Jack'}
>>> {'login': 'admin', 'password': '12345'}
```

Subscription, slicing, attribute reference

```
>>> a = [9, 8, 7, 6, 5, 4, 3, 2, 1, 0]

>>> a[5]

>>> a[:10]

>>> a.sort
```

Subscription, slicing, attribute reference

```
>>> a = [9, 8, 7, 6, 5, 4, 3, 2, 1, 0]

>>> a[5]

>>> a[:10]

>>> a.sort
>>> a.sort()
```

Subscription, slicing, attribute reference

```
>>> a = [9, 8, 7, 6, 5, 4, 3, 2, 1, 0]

>>> a[5]

>>> a[:10]

>>> a.sort
>>> a.sort()
>>> (a.sort)()
```

Including membership, identity tests

```
>>> 5 in a
>>> 5 not in a

>>> a is b
>>> a is not b
```

Comparisons

```
>>> x == y
>>> x != y

>>> x > y
>>> x >= y
>>> x < y
>>> x <= y
```

Comparisons chaining

```
>>> a <= x <= b
```

Comparisons chaining

```
>>> a <= x <= b
>>> # means (x evaluates once)
>>> a <= x and x <= b
```

Comparisons chaining

```
>>> a <= x <= b
>>> # means (x evaluates once)
>>> a <= x and x <= b

>>> holiday_start <= get_now() <= holiday_end
```

Comparisons chaining

```
>>> a <= x <= b
>>> # means (x evaluates once)
>>> a <= x and x <= b

>>> holiday_start <= get_now() <= holiday_end
```

Comparisons chaining

```
>>> a <= x <= b
>>> # means (x evaluates once)
>>> a <= x and x <= b

>>> holiday_start <= get_now() <= holiday_end

>>> get_hday_start() <= get_now() <= get_hday_end()
```

Boolean operations

```
>>> not x

>>> x and y

>>> x or y
```

Cast to bool

```
>>> bool(None)
False
>>> bool(False)
False
>>> bool(0)
False
>>> bool('')
False
>>> bool(tuple())
False
>>> bool([])
False
>>> bool(set())
False
>>> bool(frozenset())
False
>>> bool({})
False
```

```
>>> bool(True)
True
>>> bool(1)
True
>>> bool('0')
True
>>> bool((0,))
True
>>> bool([0])
True
>>> bool({0})
True
>>> bool(frozenset([0]))
True
>>> bool({0: 0})
True
```

Cast to bool

```
>>> bool(None)
False
>>> bool(False)
False
>>> bool(0)
False
>>> bool('')
False
>>> bool(tuple())
False
>>> bool([])
False
>>> bool(set())
False
>>> bool(frozenset())
False
>>> bool({})
False
```

```
>>> bool(True)
True
>>> bool(1)
True
>>> bool('0')
True
>>> bool((0,))
True
>>> bool([0])
True
>>> bool({0})
True
>>> bool(frozenset([0]))
True
>>> bool({0: 0})
True
```

Cast to bool

```
>>> obj.__bool__()
```

Cast to bool

```
>>> bool([[]])
True
```

Boolean operations

```
>>> not x

>>> x and y

>>> x or y
```

Boolean operations

```
>>> not x    == True if bool(x) is False else False

>>> x and y

>>> x or y
```

Boolean operations

```
>>> not x    == True if bool(x) is False else False

>>> x and y == x if bool(x) is False else y

>>> x or y
```

Boolean operations

```
>>> not x   == True if bool(x) is False else False

>>> x and y == x if bool(x) is False else y

>>> x or y  == x if bool(x) is True else y
```

Boolean operations

```
>>> users : list | None = get_users()
```

Boolean operations

```
>>> users : list | None = get_users()

>>> first_user : User | None = users and users[0]
```

Boolean operations

```
>>> users : list | None = get_users()
>>> first_user : User | None = users and users[0]
>>> users : list = users or []
```

Boolean operations

```
>>> email, phone, password = ...
```

Boolean operations

```
>>> email, phone, password = ...

>>> both_passed = email and phone
```

Boolean operations

```
>>> email, phone, password = ...

>>> both_passed = email and phone
>>> both_passed = bool(email and phone)
```

Boolean operations

```
>>> email, phone, password = ...

>>> both_passed = email and phone
>>> both_passed = bool(email and phone)

>>> login = email or phone
```

Boolean operations

```
>>> email, phone, password = ...

>>> both_passed = email and phone
>>> both_passed = bool(email and phone)

>>> login = email or phone
>>> password = password or generate_password()
```

Boolean operations

```
>>> email, phone, password = ...

>>> both_passed = email and phone
>>> both_passed = bool(email and phone)

>>> login = email or phone
>>> password = password or generate_password()

>>> enough_data = (email or phone) and password
```

Operator precedence

```
>>> 2 + 2 * 2 == 6
```

Operator precedence

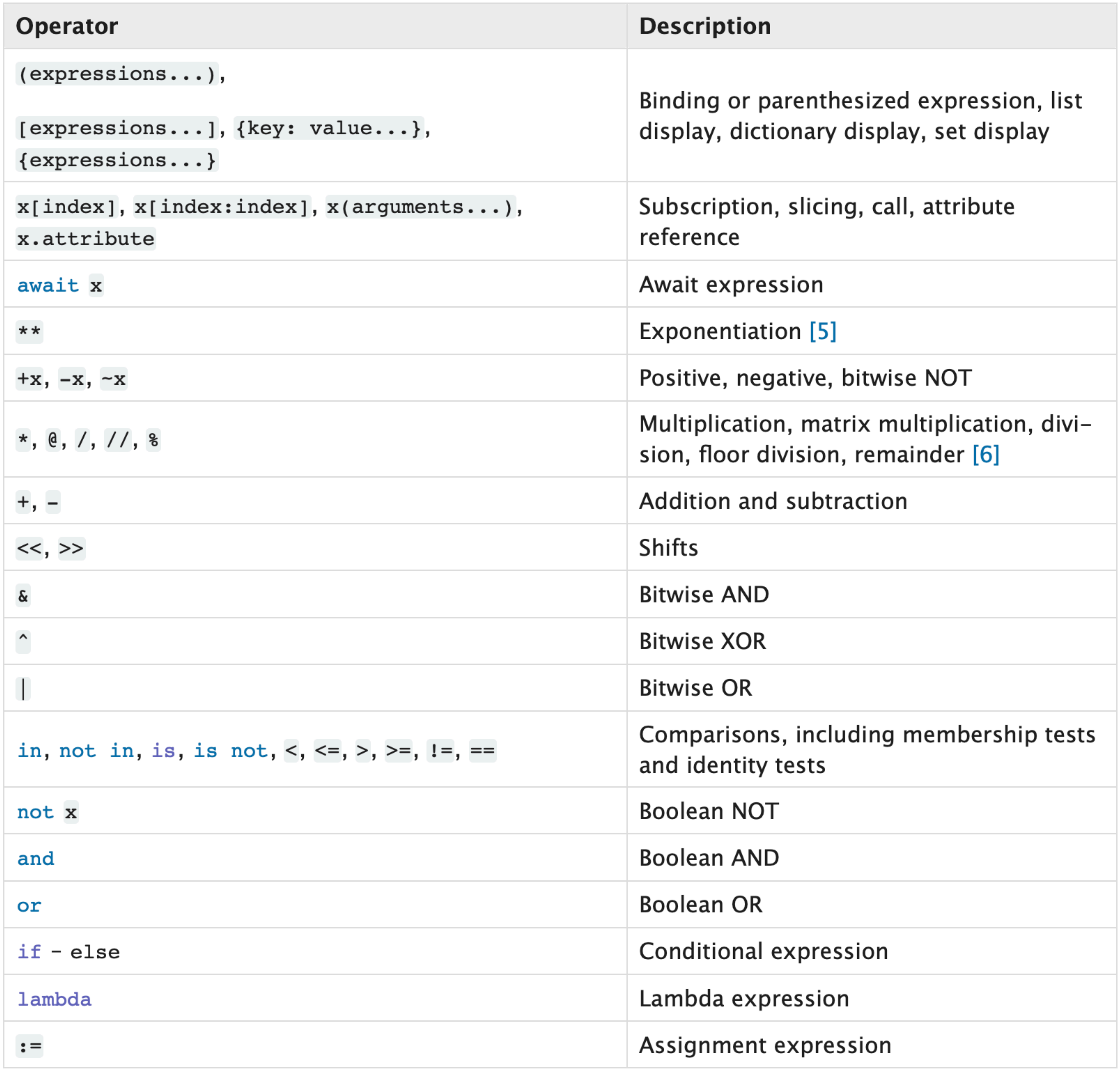
```
>>> 2 + 2 * 2 == 6

>>> a and b is True
```

Operator precedence

```
>>> 2 + 2 * 2 == 6

>>> a and b is True == (a and b) is True
# or
>>> a and b is True == a and (b is True)
```



Operator	Description
`(expressions...)`, `[expressions...]`, `{key: value...}`, `{expressions...}`	Binding or parenthesized expression, list display, dictionary display, set display
`x[index]`, `x[index:index]`, `x(arguments...)`, `x.attribute`	Subscription, slicing, call, attribute reference
`await x`	Await expression
`**`	Exponentiation [5]
`+x`, `-x`, `~x`	Positive, negative, bitwise NOT
`*`, `@`, `/`, `//`, `%`	Multiplication, matrix multiplication, division, floor division, remainder [6]
`+`, `-`	Addition and subtraction
`<<`, `>>`	Shifts
`&`	Bitwise AND
`^`	Bitwise XOR
`|`	Bitwise OR
`in`, `not in`, `is`, `is not`, `<`, `<=`, `>`, `>=`, `!=`, `==`	Comparisons, including membership tests and identity tests
`not x`	Boolean NOT
`and`	Boolean AND
`or`	Boolean OR
`if – else`	Conditional expression
`lambda`	Lambda expression
`:=`	Assignment expression

https://docs.python.org/3/reference/expressions.html#operator-precedence

Operator precedence

```
>>> 2 + 2 * 2 == 6

>>> a and b is True == (a and b) is True
# or
>>> a and b is True == a and (b is True)
```

Evaluation order

```
>>> expr1, expr2, expr3, expr4
>>> (expr1, expr2, expr3, expr4)
>>> {expr1: expr2, expr3: expr4}
>>> expr1 + expr2 * (expr3 - expr4)
>>> expr1(expr2, expr3, *expr4, **expr5)
>>> expr3, expr4 = expr1, expr2
```

https://docs.python.org/3/reference/expressions.html#evaluation-order

girafe
ai

Statements

Statements

Simple statements

- expression
- assert
- assignment
- pass
- del
- return
- yield
- raise
- break
- continue
- import
- future
- global
- nonlocal

Compound statements

- if statement
- while statement
- for statement
- try statement
- with statement
- match statement
- function defenition
- class defenition
- async with statement
- async for statement
- async function defenition

if statement

```
>>> x = int(input("Please enter an integer: "))
>>>
>>> if x < 0:
...     print('Negative integer entered')
```

if statement

```
>>> x = int(input("Please enter an integer: "))
>>>
>>> if x < 0:
...     x = 0
...     print('Negative changed to zero')
... elif x == 0:
...     print('Zero')
... elif x == 1:
...     print('Single')
... else:
...     print('More')
```

if statement

```
>>> new_name = input("Please enter a name: ")
>>> names : list | None = get_names()
>>>
>>> if names:
...     names.append(new_name)
```

if statement

```
>>> new_name = input("Please enter a name: ")
>>> names : list | None = get_names()
>>>
>>> if names:
...     names.append(new_name)


>>> if holiday_start <= get_now() <= holiday_end:
...     buy_tickets()
... else:
...     print('Try again later')
```

if statement

```
if_stmt ::=  "if" assignment_expression ":" suite
             ("elif" assignment_expression ":" suite)*
             ["else" ":" suite]
```

Conditional expression (ternary operator)

```
>>> x if condition else y
```

Boolean operations

```
>>> not x   == True if bool(x) is False else False

>>> x and y == x if bool(x) is False else y

>>> x or y  == x if bool(x) is True else y
```

Conditional expression (ternary operator)

```
>>> x if condition else y
```

Conditional expression (ternary operator)

```
>>> z = x if condition else y
```

Conditional expression (ternary operator)

```
>>> z = x if condition else y

>>> if condition:
...     z = x
... else:
...     z = y
```

Conditional expression (ternary operator)

```
>>> cold_months = [10, 11, 12, 1, 2, 3, 4]
>>> clothes = (
...     warm_clothes
...     if now().month in cold_months else
...     light_clothes
...)
```

girafe
ai

Conditional expression (ternary operator)

```
>>> z = x if x else y
```

Conditional expression (ternary operator)

```
~~>>> z = x if x else y~~
>>> z = x or y
```

Conditional expression (ternary operator)

```
~~>>> z = x if x else y~~
>>> z = x or y

>>> z = True if x else False
```

Conditional expression (ternary operator)

```
~~>>> z = x if x else y~~
>>> z = x or y

~~>>> z = True if x else False~~
>>> z = bool(x)
```

while statement

```
>>> a, b = 0, 1
>>> while a < 10:
...     print(a)
...     a, b = b, a+b
...
0
1
1
2
3
5
8
```

while statement

```
>>> while True:
...     pass
```

while statement

```
>>> while True:
...     if try_hack_pentagon():
...         break
...     print('attempt failed, trying again...')
```

while statement

```
>>> i = 0
>>> while i < 1000:
...     if try_hack_pentagon():
...         break
...     print(f'attempt {i=} failed, trying again...')
...     i += 1
... else:
...     print('all attempts failed')
```

while statement

```
>>> while True:
...     if not try_hack_pentagon():
...         print('attempt failed, trying again...')
...         continue
...     print('pentagon hacked')
...     get_reward()
...     print('let\'s try again')
```

while statement

```
while_stmt ::=  "while" assignment_expression ":" suite
                ["else" ":" suite]
```

for statement

```
>>> words = ['cat', 'window', 'defenestrate']
>>> for word in words:
...     print(word, len(word))
...
cat 3
window 6
defenestrate 12
```

for statement

```
users = {'login': 'password', 'admin': '12345',
         'cat': 'dog'}

>>> for login in users:
...     print(login)
```

for statement

```
users = {'login': 'password', 'admin': '12345',
         'cat': 'dog'}

>>> for login in users:
...     print(login)

>>> for login in users.keys():
...     print(login)

>>> for password in users.values():
...     print(password)
```

for statement

```
users = {'login': 'password', 'admin': '12345',
         'cat': 'dog'}

>>> for login in users:
...     print(login)

>>> for login in users.keys():
...     print(login)

>>> for password in users.values():
...     print(password)

>>> for login, password in users.items():
...     print(password)
```

girafe
ai

for statement

```
users = {'login': 'password', 'admin': '12345',
         'cat': 'dog'}

>>> for login, password in users.items():
...     if len(password) <= 3:
...         print(f'insecure user found: {login=}')
...         break
... else:
...     print('all users are secure')
```

for statement

```
users = {'login': 'password', 'admin': '12345',
         'cat': 'dog'}

>>> for login, password in users.items():
...     if len(password) <= 3:
...         del users[login]
```

for statement

```
users = {'login': 'password', 'admin': '12345',
         'cat': 'dog'}

>>> for login, password in users.copy().items():
...     if len(password) <= 3:
...         del users[login]
```

for statement

```
users = {'login': 'password', 'admin': '12345',
         'cat': 'dog'}

>>> for login, password in users.copy().items():
...     if len(password) <= 3:
...         del users[login]

>>> secure_users = {}
>>> for login, password in users.items():
...     if len(password) > 3:
...         secure_users[login] = password
```

for statement

```
for_stmt ::=  "for" target_list "in" expression_list ":" suite
              ["else" ":" suite]
```

range()

```
>>> for i in range(5):
...     print(i)
...
0
1
2
3
4
```

range()

```
>>> list(range(5, 10))
[5, 6, 7, 8, 9]

>>> list(range(0, 10, 3))
[0, 3, 6, 9]

>>> list(range(-10, -100, -30))
[-10, -40, -70]
```

range()

```
class range(stop)
class range(start, stop[, step])
```

range()

```
>>> r = range(10)
```

range()

```
>>> r = range(10)

>>> r
range(0, 10)
```

range()

```
>>> r = range(10)

>>> r
range(0, 10)

>>> r[5]
5
```

range()

```
>>> r = range(10)

>>> r
range(0, 10)

>>> r[5]
5

>>> r[5:]
range(5, 10)
```

range()

```
>>> r = range(10)

>>> r
range(0, 10)

>>> r[5]
5

>>> r[5:]
range(5, 10)

>>> {r[:5]: '0 to 4', range(50, 100): '50 to 99'}
{range(0, 5): '0 to 4', range(50, 100): '50 to 99'}
```

Comprehensions

```
>>> squares = []
>>> for x in range(10):
...     squares.append(x**2)
```

Comprehensions

```
>>> squares = []
>>> for x in range(10):
...     squares.append(x**2)
```

```
>>> squares = [x**2 for x in range(10)]
```

Comprehensions

```
>>> squares = []
>>> for x in range(10):
...     if x % 2 == 0
...         squares.append(x**2)
```

```
>>> squares = [x**2 for x in range(10) if x % 2 == 0]
```

Comprehensions

```
>>> words = ['intel', 'pentium', '4']

>>> alphas = []
... for word in words:
...     for char in word:
...         if char.isalpha():
...             alphas.append(char)

>>> alphas = [char for word in words for char in word if char.isalpha()]
['i', 'n', 't', 'e', 'l', 'p', 'e', 'n', 't', 'i', 'u', 'm']
```

Comprehensions

```
>>> words = ['intel', 'pentium', '4']

>>> alphas = []
... for word in words:
...     for char in word:
...         if char.isalpha():
...             alphas.append(char)

>>> alphas = [
...    char for word in words for char in word
...    if char.isalpha()
... ]
['i', 'n', 't', 'e', 'l', 'p', 'e', 'n', 't',
'i', 'u', 'm']
```

Comprehensions

```
>>> [x**2 for x in range(10) if x % 2 == 0]
[0, 4, 16, 36, 64]
```

Comprehensions

```
>>> [x**2 for x in range(10) if x % 2 == 0]
[0, 4, 16, 36, 64]

>>> {x**2 for x in range(10) if x % 2 == 0}
{0, 64, 4, 36, 16}
```

Comprehensions

```
>>> [x**2 for x in range(10) if x % 2 == 0]
[0, 4, 16, 36, 64]

>>> {x**2 for x in range(10) if x % 2 == 0}
{0, 64, 4, 36, 16}

>>> {x: x**2 for x in range(10) if x % 2 == 0}
{0: 0, 2: 4, 4: 16, 6: 36, 8: 64}
```

```
>>> squares = (x**2 for x in range(10*100))
```

Generator expression

```
>>> squares = (x**2 for x in range(10*100))
```

Generator expression

```
>>> squares = (x**2 for x in range(10*100))
>>> squares
<generator object <genexpr> at 0x10c853290>
```

Generator expression

```
>>> test = [print(x**2) for x in range(3)]
>>> print(test)
>>> list(test)

>>> test = (print(x**2) for x in range(3))
>>> print(test)
>>> list(test)
```

girafe ai

Generator expression

```
>>> test = [print(x**2) for x in range(3)]
0
1
4
>>> print(test)
[None, None, None]
>>> list(test)
[None, None, None]
```

Generator expression

```
>>> test = [print(x**2) for x in range(3)]
0
1
4
>>> print(test)
[None, None, None]
>>> list(test)
[None, None, None]

>>> test = (print(x**2) for x in range(3))
>>> print(test)
<generator object <genexpr> at 0x10c9e7140>
>>> list(test)
0
1
4
[None, None, None]
```

girafe
ai

Recent improvements of expressions and statements

3.6: Formatted string literals (f-strings)

```
>>> name = input('Please, type your name: ')
Please, type your name: Jack
>>> print(f'Hello, {name}!')
Hello, Jack!
```

3.8: Formatted string literals (f-strings)

```
>>> a, b = 0, 1
>>> while a < 10:
...     print(f'Current state: {a=}, {b=}')
...     a, b = b, a+b
...
Current state: a=0, b=1
Current state: a=1, b=1
Current state: a=1, b=2
Current state: a=2, b=3
Current state: a=3, b=5
Current state: a=5, b=8
Current state: a=8, b=13
```

3.8: Assignment expressions (walrus operator)

```python
if (n := len(a)) > 10:
    print(f"List is too long "
          f"({n} elements, expected <= 10)")
```

3.9: Dictionary Merge & Update Operators

```
>>> x = {"key1": "1 from x", "key2": "2 from x"}
>>> y = {"key2": "2 from y", "key3": "3 from y"}
```

3.9: Dictionary Merge & Update Operators

```
>>> x = {"key1": "1 from x", "key2": "2 from x"}
>>> y = {"key2": "2 from y", "key3": "3 from y"}

>>> z = x | y
>>> z
{'key1': '1 from x', 'key2': '2 from y', 'key3': '3 from y'}
```

3.9: Dictionary Merge & Update Operators

```
>>> x = {"key1": "1 from x", "key2": "2 from x"}
>>> y = {"key2": "2 from y", "key3": "3 from y"}

>>> z = x | y  # z = {**x, **y}
>>> z
{'key1': '1 from x', 'key2': '2 from y', 'key3': '3 from y'}
```

3.9: Dictionary Merge & Update Operators

```
>>> x = {"key1": "1 from x", "key2": "2 from x"}
>>> y = {"key2": "2 from y", "key3": "3 from y"}

>>> z = x | y  # z = {**x, **y}
>>> z
{'key1': '1 from x', 'key2': '2 from y', 'key3': '3 from y'}

>>> x |= y
>>> x
{'key1': '1 from x', 'key2': '2 from y', 'key3': '3 from y'}
```

3.9: Dictionary Merge & Update Operators

```
>>> x = {"key1": "1 from x", "key2": "2 from x"}
>>> y = {"key2": "2 from y", "key3": "3 from y"}

>>> z = x | y  # z = {**x, **y}
>>> z
{'key1': '1 from x', 'key2': '2 from y', 'key3': '3 from y'}

>>> x |= y  # x.update(y)
>>> x
{'key1': '1 from x', 'key2': '2 from y', 'key3': '3 from y'}
```

3.10: Structural Pattern Matching

```
match subject:
    case <pattern_1>:
        <action_1>
    case <pattern_2>:
        <action_2>
    case <pattern_3>:
        <action_3>
    case _:
        <action_wildcard>
```

3.10: Structural Pattern Matching

```
match status:
    case 400:
        return "Bad request"
    case 404:
        return "Not found"
    case 418:
        return "I'm a teapot"
    case _:
        return "Something's wrong with the internet"
```

3.10: Structural Pattern Matching

```
match status:
    case 400:
        return "Bad request"
    case 404:
        return "Not found"
    case 418:
        return "I'm a teapot"
    case 401 | 403 | 404:
        return "Not allowed"
    case _:
        return "Something's wrong with the internet"
```

3.10: Structural Pattern Matching

```
# point is an (x, y) tuple
match point:
    case (0, 0):
        print("Origin")
    case (0, y):
        print(f"Y={y}")
    case (x, 0):
        print(f"X={x}")
    case (x, y):
        print(f"X={x}, Y={y}")
    case _:
        raise ValueError("Not a point")
```

3.10: Structural Pattern Matching

```
match points:
    case []:
        print("No points in the list.")
    case [Point(0, 0)]:
        print("The origin is the only point in the list.")
    case [Point(x, y)]:
        print(f"A single point {x}, {y} is in the list.")
    case [Point(0, y1), Point(0, y2)]:
        print(f"Two points on the Y axis at {y1}, {y2} "
              f"are in the list.")
    case _:
        print("Something else is found in the list.")
```

3.10: Structural Pattern Matching

```
match test_variable:
    case ('warning', code, 40):
        print("A warning has been received.")
    case ('error', code, _):
        print(f"An error {code} occurred.")
```

3.10: Structural Pattern Matching

```
match point:
    case (x, y) if x == y:
        print(f"The point is on the diagonal Y=X at {x}.")
    case (x, y) as p:
        print(f"Point {p} is not on the diagonal.")
```

girafe
ai