

# The \*Best Python Cheat Sheet

*Just what you need*

Built-in (1)	Execution (25)	List (10)	Set (11)
Bytes (12)	Flow control (5)	Number (21)	String (17)
Class (13)	Function (12)	Operator (3)	Test (25)
Debug (25)	Generator (16)	Regex (18)	Time (22)
Decorator (15)	Iterator (16)	Resource (25)	Tuple (9)
Dictionary (10)	Keyword (1)	Scope (7)	Types (25)
Exception (23)	Library (25)	Sequence (8)	

## Keyword

and	del	global	nonlocal	type <sup>0</sup>
as	elif	if	not	while
assert	else	import	or	with
break	except	in	pass	yield
case <sup>0</sup>	False	is	raise	_ <sup>0</sup>
class	finally	lambda	return	
continue	for	match <sup>0</sup>	True	
def	from	None	try	

<sup>0</sup>Soft keywords

## Built-in

abs(number)	Absolute value of <a href="#">number</a>	bytes(...)	New <a href="#">bytes</a> object from byte-integers, string, bytes
aiter(async_iterable)	Asynchronous iterator for an asynchronous iterable	callable(object)	True if object is callable
all(iterable)	True if all elements of iterable are true (all([]) == True)	chr(i)	One character string for unicode ordinal i (0 <= i <= 0x10ffff)
any(iterable)	True if any element of iterable is true (any([]) == False)	classmethod(func)	Transform function into <a href="#">class method</a>
ascii(object)	A string with a printable representation of an object	compile(source, ...)	Compile source into code or AST object
bin(number)	Convert integer number to binary string	complex(real=0, imag=0)	<a href="#">Complex number</a> with the value real + imag*1j
bool(object)	Boolean value	delattr(object, name)	Delete the named attribute, if object allows
breakpoint(*args, **kwds)	Drop into debugger via sys.breakpointhook(*args, **kwds)	dict(...)	Create new <a href="#">dictionary</a>
bytearray(...)	New array of <a href="#">bytes</a> from byte-integers, string, bytes, object with buffer API	dir([object])	List of names in the local scope, or object.__dir__() or attributes
		divmod(x, y)	Return (quotient x//y, remainder x%y)

<code>enumerate(iterable, start=0)</code>	Enumerate object as (n, item) pairs with n initialised to start value	<code>issubclass(cls, cls_or_tuple)</code>	True if class is subclass of given class(es)
<code>eval(source, globals=None, locals=None)</code>	Execute Python expression, string or code object from compile()	<code>iter(object, ...)</code>	<u>Iterator</u> for object
<code>exec(source, globals=None, locals=None)</code>	Execute Python statements, string or code object from compile()	<code>len(object)</code>	Length of object
<code>filter(func, iterable)</code>	Iterator yielding items where func(item) is true, or bool(item) if func is None	<code>list(...)</code>	Create <u>list</u>
<code>float(x=0)</code>	<u>Floating point number</u> from number or string	<code>locals()</code>	Dictionary of current local symbol table
<code>format(object, format_spec='')</code>	Formatted representation	<code>map(func, *iterables)</code>	Apply function to every item of iterable(s)
<code>frozenset(...)</code>	New <u>frozenset</u> object	<code>max(..., key=func)</code>	Largest item of iterable or arguments, optional key function extracts value
<code>getattr(object, name[, default])</code>	Get value of named attribute of object, else default or raise exception	<code>memoryview(object)</code>	Access internal object data via buffer protocol
<code>globals()</code>	Dictionary of current module namespace	<code>min(..., key=func)</code>	Smallest item of iterable or arguments, optional key function extracts value
<code>hasattr(object, name)</code>	True if object has named attribute	<code>next(iterator[, default])</code>	Next item from <u>iterator</u> , optionally return default instead of StopIteration
<code>hash(object)</code>	Hash value of object (see object.__hash__())	<code>object()</code>	New featureless object
<code>help(...)</code>	Built-in help system	<code>oct(number)</code>	Convert integer to octal string
<code>hex(number)</code>	Convert integer to lowercase hexadecimal string	<code>open(file, ...)</code>	Open file object
<code>id(object)</code>	Return unique integer identifier of object	<code>ord(chr)</code>	Integer representing Unicode code point of character
<code>__import__(name, ...)</code>	Invoked by the import statement	<code>pow(base, exp, mod=None)</code>	Return <i>base</i> to the power of <i>exp</i>
<code>input(prompt='')</code>	Read string from stdin, with optional prompt	<code>print(value, ...)</code>	Print object to text stream file
<code>int(...)</code>	Create <u>integer</u> from number or string	<code>property(...)</code>	<u>Property</u> decorator
<code>isinstance(object, cls_or_tuple)</code>	True if object is instance of given class(es)	<code>range(...)</code>	Generate integer sequence
		<code>repr(object)</code>	String representation of object for debugging

<code>reversed(sequence)</code>	Reverse iterator	<code>sum(iterable, start=0)</code>	Sums items of iterable, optionally adding start value
<code>round(number, ndigits=None)</code>	<u>Number</u> rounded to ndigits precision after decimal point	<code>super(...)</code>	Proxy object that delegates method calls to parent or sibling
<code>set(...)</code>	New <u>set</u> object	<code>tuple(iterable)</code>	Create a <u>tuple</u>
<code>setattr(object, name, value)</code>	Set object attribute value by name	<code>type(...)</code>	Type of an object, or build new type
<code>slice(...)</code>	Slice object representing a set of <u>indices</u>	<code>vars([object])</code>	Return object.__dict__ or locals() if no argument
<code>sorted(iterable, key=None, reverse=False)</code>	New sorted list from the items in iterable	<code>zip(*iterables, strict=False)</code>	Iterate over multiple iterables in parallel, strict requires equal length
<code>staticmethod(func)</code>	Transform function into <u>static method</u>		
<code>str(...)</code>	<u>String</u> description of object		

## Operator

Precedence (high->low)	Description
<code>(...)</code> <code>[...]</code> <code>{...}</code> <code>{...:...}</code>	<u>tuple</u> , <u>list</u> , <u>set</u> , <u>dict</u>
<code>s[i]</code> <code>s[i:j]</code> <code>s.attr</code> <code>f(...)</code>	<u>index</u> , <u>slice</u> , attribute, <u>function call</u>
<code>await x</code>	await expression
<code>+x</code> , <code>-x</code> , <code>~x</code>	unary positive, negative, bitwise NOT
<code>x ** y</code>	power
<code>x * y</code> , <code>x @ y</code> , <code>x / y</code> , <code>x // y</code> , <code>x % y</code>	multiply, maxtrix multiply, divide, floor divide, modulus
<code>x + y</code> , <code>x - y</code>	add, subtract
<code>x &lt;&lt; y</code> <code>x &gt;&gt; y</code>	bitwise shift left, right
<code>x &amp; y</code>	bitwise and
<code>x ^ y</code>	bitwise exclusive or
<code>x   y</code>	bitwise or
<code>x &lt; y</code> <code>x &lt;= y</code> <code>x &gt; y</code> <code>x &gt;= y</code> <code>x == y</code> <code>x != y</code> <code>x is y</code> <code>x is not y</code> <code>x in s</code> <code>x not in s</code>	<u>comparison</u> , identity, <u>membership</u>
<code>not x</code>	boolean negation
<code>x and y</code>	boolean and
<code>x or y</code>	boolean or
<code>... if ... else ...</code>	<u>conditional expression</u>
<code>lambda</code>	lambda expression
<code>:=</code>	<u>assignment expression</u>

Assignment	Usually equivalent
<code>a = b</code>	Assign object b to label a
<code>a += b</code>	<code>a = a + b</code>
<code>a -= b</code>	<code>a = a - b</code>
<code>a *= b</code>	<code>a = a * b</code>
<code>a /= b</code>	<code>a = a / b</code> (true division)
<code>a //= b</code>	<code>a = a // b</code> (floor division)
<code>a %= b</code>	<code>a = a % b</code>
<code>a **= b</code>	<code>a = a ** b</code>
<code>a &amp;= b</code>	<code>a = a &amp; b</code>
<code>a  = b</code>	<code>a = a   b</code>
<code>a ^= b</code>	<code>a = a ^ b</code>
<code>a &gt;&gt;= b</code>	<code>a = a &gt;&gt; b</code>
<code>a &lt;&lt;= b</code>	<code>a = a &lt;&lt; b</code>

**Assignment expression**

Assign a value and return that value using the *walrus operator*.

```
if matching := pattern.search(data):
    do_something(matching)

count = 0
while (count := count + 1) < 5:
    print(count)

>>> z = [1, 2, 3, 4, 5]
>>> [x for i in z if (x:=i**2) > 10]
[16, 25]
```

**Assignment unpacking**

Unpack multiple values to a name using the *splat operator*.

```
head, *body      = s      # assign first value of s to head, remainder to body
head, *body, tail = s      # assign first and last values of s to head and tail,
remainder to body
*body, tail      = s      # assign last value of s to tail, remainder to body
s = [*iterable[, ...]]    # unpack to list
s = (*iterable[, ...])    # unpack to tuple
s = {*iterable[, ...]}    # unpack to set
d2 = {**d1[, ...]}        # unpack to dict
```

**Flow control**

```
for item in <iterable>:
    ...
[else:
    ...]                # if loop completes without break

while <condition>:
    ...
[else:
    ...]                # if loop completes without break

break                  # immediately exit loop
continue               # skip to next loop iteration
return[ value]         # exit function, return value | None
yield[ value]          # exit generator, yield value | None
assert <expr>[, message] # if not expression raise AssertionError(message)
```

```
if condition:
    ...
[elif condition:
    ...]*
[else:
    ...]

<expression1> if <condition> else <expression2>

with <expression>[ as name]:
    ...
```

**Match**

3.10+

```

match <expression>:
    case <pattern>[ if <condition>]:
        ...
    case <pattern1> | <pattern2>:      # OR pattern
        ...
    case _:                          # default case
        ...

```

**Match case pattern**

1/'abc' /True/None/math.pi	Value pattern, match literal or dotted name
<type>()	Class pattern, match any object of that type
<type>(<name>=<pattern>, ...)	Class pattern, match object with matching attributes
<name>	Capture pattern, match any object, bind to name
_	Wildcard, match any object
<pattern>   <pattern> [  ...]	Or pattern, match any of the patterns
<pattern> as <name>	As pattern, bind match to name
[<pattern>[, ...[, *args]]	Sequence pattern (list tuple) matches sequence with matching items
{<value_pattern>: <pattern>[, ...[, **kwds]]}	Mapping pattern matches any dictionary with matching items

- Class patterns **do not** create a new instance of the class
- Patterns can be bracketed to override precedence [| > as > ,]
- Built-in types allow a single positional pattern that is matched against the entire object.
- Names bound in the matching case + names bound in its block are visible after the match statement

**Context manager**

A *with* statement takes an object with special methods:

- `__enter__()` - locks resources and optionally returns an object
- `__exit__()` - releases resources, handles an exception raised in the block, optionally suppressing it by returning True

```

class MyOpen:
    def __init__(self, filename):
        self.filename = filename
    def __enter__(self):
        self.file = open(self.filename)
        return self.file
    def __exit__(self, exc_type, exception, traceback):
        self.file.close()

>>> with open('test.txt', 'w') as file: ...
...     file.write('Hello World!')
>>> with MyOpen('test.txt') as file: ...
...     print(file.read())
Hello World!

```

## Scope

Scope levels:

Builtin	Names pre-assigned in <i>builtins</i> module	Generator expression	Names contained within generator expression
Module (global)	Names defined in current module Code in global scope cannot access local variables	Comprehension	Names contained within comprehension
Enclosing (closure)	Names defined in any enclosing functions	Class	Names shared across all instances
Function (local)	Names defined in current function By default, has read-only access to module and enclosing function names By default, assignment creates a new local name <i>global &lt;name&gt;</i> grants read/write access to specified module name <i>nonlocal &lt;name&gt;</i> grants read/write access to specified name in closest enclosing function defining that name	Instance	Names contained within a specific instance
		Method	Names contained within a specific instance method

■ *globals()* - return *dict* of module scope variables

■ *locals()* - return *dict* of local scope variables

```

>>> global_variable = 1
>>> def read_global():
...     print(global_variable)
...     local_variable = "only available in this function"
...     print(local_variable)
>>> read_global()
1

>>> def write_global():
...     global global_variable
...     global_variable = 2
>>> write_global()
>>> print(global_variable)
2

>>> def write_nonlocal():
...     x = 1
...     def nested():
...         nonlocal x
...         x = 2
...     nested()
...     print(x)
>>> write_nonlocal()
2

>>> class C:
...     class_variable = 1
...     def __init__(self):
...         self.instance_variable = 2
...     def method(self):
...         self.instance_variable = 3
...         C.class_variable = 3
...         method_variable = 1

```

## Sequence

Operations on sequence types (Bytes, List, Tuple, String).

<code>x in s</code>	True if any <code>s[i]==x</code> # ?? __contains__? is == right op	<code>max(s)</code>	Largest item
<code>x not in s</code>	True if no <code>s[i]==x</code>	<code>s.index(x[, start[, stop]])</code>	Smallest i where <code>s[i]==x</code> , start/stop bounds search
<code>s1 + s2</code>	Concatenate s1 and s2	<code>reversed(s)</code>	Iterator on s in reverse order (for string use <code>reversed(list(s))</code> )
<code>s*n, n*s</code>	Concatenate n copies of s	<code>sorted(s1, cmp=func, key=getter, reverse=False)</code>	New sorted list
<code>s.count(x)</code>	Count of <code>s[i]==x</code>		
<code>len(s)</code>	Number of items		
<code>min(s)</code>	Smallest item		

## Indexing

Select items from sequence by index or slice.



```

>>> s = [0, 1, 2, 3, 4]
>>> s[0]           # 0-based indexing
0
>>> s[-1]          # negative indexing from end
4
>>> s[slice(2)]     # slice(stop) - index until stop (exclusive)
[0, 1]
>>> s[slice(1, 5, 3)] # slice(start, stop[, step]) - index from start to stop
(exclusive), with optional step size (+|-)
[1, 4]
>>> s[:2]          # slices are created implicitly when indexing with ':'
[start:stop:step]
[0, 1]
>>> s[3::-1]       # negative steps
[3, 2, 1, 0]
>>> s[1:3]
[1, 2]
>>> s[1:5:2]
[1, 3]

```

### Comparison

- Sequence comparison: values are compared in order until a pair of unequal values is found. The comparison of these two values is then returned. If all values are equal, the shorter sequence is lesser.
- A sortable class should define `__eq__()`, `__lt__()`, `__gt__()`, `__le__()` and `__ge__()` comparison special methods.
- With `functools @total_ordering` decorator a class need only provide `__eq__()` and one other comparison special method.

```

from functools import total_ordering

@total_ordering
class C:
    def __init__(self, a):
        self.a = a
    def __eq__(self, other):
        if isinstance(other, type(self)):
            return self.a == other.a
        return NotImplemented
    def __lt__(self, other):
        if isinstance(other, type(self)):
            return self.a < other.a
        return NotImplemented

```

### Tuple

Immutable hashable sequence.

<code>s = (1, 'a', 3.0)</code>	Create tuple
<code>s = 1, 'a', 3.0</code>	
<code>s = (1,)</code>	Single-item tuple
<code>s = ()</code>	Empty tuple
<code>(1, 2, 3) == (1, 2) + (3,)</code>	Add makes new tuple
<code>(1, 2, 1, 2) == (1, 2) * 2</code>	Multiply makes new tuple

**Named tuple**

Subclass with named items.

```
>>> from collections import namedtuple
>>> Point = namedtuple('Point', ('x', 'y')) # or namedtuple('Point', 'x y')
>>> p = Point(1, y=2)
Point(x=1, y=2)
>>> p[0]
1
>>> p.y
2
```

**List**

Mutable non-hashable sequence.

s = [1, 'a', 3.0]	Create list	s.extend(it)	Add elements from iterable to end
s = list(range(3))		s[len(s):len(s)] = it	
s[i] = x	Replace item index i with x	s.insert(i, x)	Insert item at index i
s[<slice>] = it	Replace slice with iterable	s[i:i] = [x]	
del s[<slice>]	Delete slice	s.remove(x)	Remove item
s[<slice>] = []		del s[s.index(x)]	
s.append(x)	Add element to end	y = s.pop([i])	Remove and return last item, or indexed item
s += x		s.reverse()	Reverse in place
s[len(s):len(s)] = [x]		s.sort(cmp=func, key=getter, reverse=False)	Sort in place, default ascending

**List comprehension**

```
result = [expression for item1 in iterable1{ if condition1}
           {for item2 in iterable2{ if condition2} ... for itemN in iterableN{ if conditionN}}]

# is equivalent to:

result = []
for item1 in iterable1:
    for item2 in iterable2:
        ...
        for itemN in iterableN:
            if condition1 and condition2 ... and conditionN:
                result.append(expression)
```

**Dictionary**

Mutable non-hashable key:value pair mapping.

dict()	Empty dict	dict(zip(keys, values))	Create from sequences of keys and values
dict(<sequence mapping>)	Create from key:value pairs	dict.fromkeys(keys, value=None)	Create from keys, all set to value
dict(**kwds)	Create from keyword arguments	d.keys()	Iterable of keys

d.values()	Iterable of values	d.clear()	Remove all items
d.items()	Iterable of (key, value) pairs	d.copy()	Shallow copy
d.get(key, default=None)	Get value for key, or default	collections.defaultdict(<type>)	dict with default value <type>()
d.setdefault(key, default=None)	Get value for key, add if missing	collections.defaultdict(lambda: 42)	e.g. dict with default value 42
d.pop(key)	Remove and return value for key, raise KeyError if missing	d1.update(d2)	Add/replace key:value pairs from d2 to d1
d.popitem()	Remove and return (key, value) pair (last-in, first-out)	d1  = d2 <b>3.9+</b>	
		d3 = d1   d2 <b>3.9+</b>	Merge to new dict, d2 trumps d1
		d3 = {**d1, **d2}	
		{k for k, v in d.items() if v==value}	Set of keys with given value

**Dict comprehension**

```
# {k: v for k, v in <iterable>[ if <condition>]}
```

```
>>> {x: x**2 for x in (2, 4, 6) if x < 5}
{2: 4, 4: 16}
```

**Set**

Mutable (*set*) and immutable (*frozenset*) sets.

set(iterable=None)	New set from iterable, or empty	s1.intersection(s2[, s3...])	New set of shared elements
frozenset(iterable=None)	But {} creates an empty dictionary (sad!)	s1 & s2	
len(s)	Cardinality	s1.intersection_update(s2) [ <i>mutable</i> ]	Update elements to intersection with s2
v in s	Test membership	s1.union(s2[, s3...])	New set of all elements
v not in s		s1   s2	
s1.issubset(s2)	True if s1 is subset of s2	s1.difference(s2[, s3...])	New set of elements unique to s1
s1.issuperset(s2)	True if s1 is superset of s2	s1 - s2	
s.add(v) [ <i>mutable</i> ]	Add element	s1.difference_update(s2) [ <i>mutable</i> ]	Remove elements intersecting with s2
s.remove(v) [ <i>mutable</i> ]	Remove element (KeyError if not found)	s1.symmetric_difference(s2)	New set of unshared elements
s.discard(v) [ <i>mutable</i> ]	Remove element if present	s1 ^ s2	
s.pop() [ <i>mutable</i> ]	Remove and return arbitrary element (KeyError if empty)	s1.symmetric_difference_update(s2) [ <i>mutable</i> ]	Update elements to symmetric difference with s2
s.clear() [ <i>mutable</i> ]	Remove all elements	s.copy()	Shallow copy
		s.update(it1[, it2...]) [ <i>mutable</i> ]	Add elements from iterables

**Set comprehension**

```
# {x for x in <iterable>[ if <condition>]}
```

```
>>> {x for x in 'abracadabra' if x not in 'abc'}
{'r', 'd'}
```

## Bytes

Immutable sequence of bytes. Mutable version is bytearray.

<code>b'&lt;str&gt;'</code>	Create from ASCII characters and <code>\x00-\xff</code>	<code>&lt;bytes&gt; = &lt;bytes&gt;[&lt;slice&gt;]</code>	Return <i>bytes</i> even if only one element
<code>bytes(&lt;ints&gt;)</code>	Create from int sequence	<code>list(&lt;bytes&gt;)</code>	Return ints in range 0 to 255
<code>bytes(&lt;str&gt;, 'utf-8')</code> <code>&lt;str&gt;.encode('utf-8')</code>	Create from string	<code>&lt;bytes_sep&gt;.join(&lt;byte_objs&gt;)</code>	Join <i>byte_objs</i> sequence with <i>bytes_sep</i> separator
<code>&lt;int&gt;.to_bytes(1, order, signed=False)</code>	Create from int (order='big' 'little')	<code>str(&lt;bytes&gt;, 'utf-8')</code> <code>&lt;bytes&gt;.decode('utf-8')</code>	Convert bytes to string
<code>bytes.fromhex('&lt;hex&gt;')</code>	Create from hex pairs (can be separated by whitespace)	<code>int.from_bytes(b'bytes', order, signed=False)</code>	Return int from bytes (order='big' 'little')
<code>&lt;int&gt; = &lt;bytes&gt;[&lt;index&gt;]</code>	Return int in range 0 to 255	<code>&lt;bytes&gt;.hex(sep='', bytes_per_sep=2)</code>	Return hex pairs

```
def read_bytes(filename):
    with open(filename, 'rb') as file:
        return file.read()

def write_bytes(filename, bytes_obj):
    with open(filename, 'wb') as file:
        file.write(bytes_obj)
```

## Function

### Function definition

```
def f(*args): ...           # f(1, 2, 3)
def f(x, *args): ...        # f(1, 2, 3)
def f(*args, z): ...        # f(1, 2, z=3)

def f(**kwargs): ...        # f(x=1, y=2, z=3)
def f(x, **kwargs): ...      # f(x=1, y=2, z=3) | f(1, y=2, z=3)
def f(*args, **kwargs): ...  # f(x=1, y=2, z=3) | f(1, y=2, z=3) | f(1, 2, z=3) | f(1, 2, 3)
def f(x, *args, **kwargs): ... # f(x=1, y=2, z=3) | f(1, y=2, z=3) | f(1, 2, z=3) | f(1, 2, 3)
def f(*args, y, **kwargs): ... # f(x=1, y=2, z=3) | f(1, y=2, z=3)

def f(*, x, y, z): ...       # f(x=1, y=2, z=3)
def f(x, *, y, z): ...       # f(x=1, y=2, z=3) | f(1, y=2, z=3)
def f(x, y, *, z): ...       # f(x=1, y=2, z=3) | f(1, y=2, z=3) | f(1, 2, z=3)
```

### Function call

```
args = (1, 2)                # * expands sequence to positional arguments
kwargs = {'x': 3, 'y': 4}    # ** expands dictionary to keyword arguments
func(*args, **kwargs)        # is the same as:
func(1, 2, x=3, y=4)
```

**Class****Instantiation**

```
class C:
    def __init__(self, a):
        self.a = a
    def __repr__(self):
        """Used for repr(c), also for str(c) if __str__ not defined."""
        return f'{self.__class__.__name__}({self.a!r})'
    def __str__(self):
        return str(self.a)

    @classmethod
    def get_class_name(cls): # passed class rather than instance
        return cls.__name__

    @staticmethod
    def static(): # passed nothing
        return 1

# class instantiation does this
obj = cls.__new__(cls, *args, **kwds)
if isinstance(obj, cls):
    obj.__init__(*args, **kwds)
```

**Instance property**

```
class C:
    @property
    def f(self):
        if not hasattr(self, '_f'):
            return
        return self._f
    @f.setter
    def f(self, value):
        self._f = value
```

**Class special methods**

Operator	Method
self + other	<code>__add__(self, other)</code>
other + self	<code>__radd__(self, other)</code>
self += other	<code>__iadd__(self, other)</code>
self - other	<code>__sub__(self, other)</code>
other - self	<code>__rsub__(self, other)</code>
self -= other	<code>__isub__(self, other)</code>
self * other	<code>__mul__(self, other)</code>
other * self	<code>__rmul__(self, other)</code>
self *= other	<code>__imul__(self, other)</code>
self @ other	<code>__matmul__(self, other)</code>
other @ self	<code>__rmatmul__(self, other)</code>
self @= other	<code>__imatmul__(self, other)</code>
self / other	<code>__truediv__(self, other)</code>
other / self	<code>__rtruediv__(self, other)</code>
self /= other	<code>__itruediv__(self, other)</code>
self // other	<code>__floordiv__(self, other)</code>
other // self	<code>__rfloordiv__(self, other)</code>
self //= other	<code>__ifloordiv__(self, other)</code>
self % other	<code>__mod__(self, other)</code>
other % self	<code>__rmod__(self, other)</code>
self %= other	<code>__imod__(self, other)</code>
self ** other	<code>__pow__(self, other)</code>
other ** self	<code>__rpow__(self, other)</code>
self **= other	<code>__ipow__(self, other)</code>
self << other	<code>__lshift__(self, other)</code>
other << self	<code>__rlshift__(self, other)</code>
self <<= other	<code>__ilshift__(self, other)</code>
self >> other	<code>__rshift__(self, other)</code>
other >> self	<code>__rrshift__(self, other)</code>
self >>= other	<code>__irshift__(self, other)</code>
self & other	<code>__and__(self, other)</code>
other & self	<code>__rand__(self, other)</code>
self &= other	<code>__iand__(self, other)</code>
self   other	<code>__or__(self, other)</code>
other   self	<code>__ror__(self, other)</code>
self  = other	<code>__ior__(self, other)</code>
self ^ other	<code>__xor__(self, other)</code>
other ^ self	<code>__rxor__(self, other)</code>
self ^= other	<code>__ixor__(self, other)</code>
<code>divmod(self, other)</code>	<code>__divmod__(self, other)</code>
<code>divmod(self, other)</code>	<code>__rdivmod__(self, other)</code>

Operator	Method
-self	<code>__neg__(self)</code>
+self	<code>__pos__(self)</code>
<code>abs(self)</code>	<code>__abs__(self)</code>
~self	<code>__invert__(self)</code> [bitwise]
<code>self == other</code>	<code>__eq__(self)</code> [default 'is', requires <code>__hash__</code> ]
<code>self != other</code>	<code>__ne__(self)</code>
<code>self &lt; other</code>	<code>__lt__(self, other)</code>
<code>self &lt;= other</code>	<code>__le__(self, other)</code>
<code>self &gt; other</code>	<code>__gt__(self, other)</code>
<code>self &gt;= other</code>	<code>__ge__(self, other)</code>
<code>item in self</code>	<code>__contains__(self, item)</code>
<code>bool(self)</code>	<code>__bool__(self)</code>
<code>bytes(self)</code>	<code>__bytes__(self)</code>
<code>complex(self)</code>	<code>__complex__(self)</code>
<code>float(self)</code>	<code>__float__(self)</code>
<code>int(self)</code>	<code>__int__(self)</code>
<code>round(self)</code>	<code>__round__(self[, ndigits])</code>
<code>math.ceil(self)</code>	<code>__ceil__(self)</code>
<code>math.floor(self)</code>	<code>__floor__(self)</code>
<code>math.trunc(self)</code>	<code>__trunc__(self)</code>
<code>dir(self)</code>	<code>__dir__(self)</code>
<code>format(self)</code>	<code>__format__(self, format_spec)</code>
<code>hash(self)</code>	<code>__hash__(self)</code>
<code>iter(self)</code>	<code>__iter__(self)</code>
<code>len(self)</code>	<code>__len__(self)</code>
<code>repr(self)</code>	<code>__repr__(self)</code>
<code>reversed(self)</code>	<code>__reversed__(self)</code>
<code>str(self)</code>	<code>__str__(self)</code>
<code>self(*args, **kwds)</code>	<code>__call__(self, *args, **kwds)</code>
<code>self[...]</code>	<code>__getitem__(self, key)</code>
<code>self[...] = 1</code>	<code>__setitem__(self, key, value)</code>
<code>del self[...]</code>	<code>__delitem__(self, key)</code>
<code>other[self]</code>	<code>__index__(self)</code>
<code>self.name</code>	<code>__getattr__(self, name)</code> <code>__getattribute__(self, name)</code> [if <code>AttributeError</code> ]
<code>self.name = 1</code>	<code>__setattr__(self, name, value)</code>
<code>del self.name</code>	<code>__delattr__(self, name)</code>
<code>with self:</code>	<code>__enter__(self)</code> <code>__exit__(self, exc_type, exc_value, traceback)</code>
<code>await self</code>	<code>__await__(self)</code>

## Decorator

A decorator is a callable that manipulates and returns a function.

```
# wraps decorator copies metadata of decorated function (func) to wrapped function
(out)
from functools import wraps

def show_call(func):
    """
    Print function name and arguments each time it is called.
    """
    @wraps(func)
    def out(*args, **kwargs):
        print(func.__name__, args, kwargs)
        return func(*args, **kwargs)
    return out

@show_call
def add(x, y):
    return x + y
```

## Iterator

An iterator implements the `__iter__()` method, returning an iterable that implements the `__next__()` method. The `__next__()` method returns the next item in the collection and raises `StopIteration` when done.

```
def IterableIterator:
    def __iter__(self):
        """Make class iterable."""
        return self

    def __next__(self):
        """Implement to be iterable."""
        if at_the_end:
            raise StopIteration
        return next_item

c = IterableIterator()
it = iter(c) # get iterator
next(it)    # get next item
while value := next(it):
    print(value)
```

## Generator

A function with a `yield` statement returns a generator iterator and suspends function processing. Each iteration over the generator iterator resumes function execution, returns the next yield value, and suspends again.



```
def gen():
    """Generator function"""
    for i in range(10):
        yield i
g = gen()

g = (expression for item in iterable if condition) # generator expression

next(g)          # next item
for i in g: ...   # iterator over items
list(g)          # list all items
yield from g      # delegate yield to another generator
```

## String

Immutable sequence of characters.

<substring> in s	True if string contains <i>substring</i>	s.find(<substring>)	Index of first match or -1
s.startswith(<prefix>[, start[, end]])	True if string starts with prefix, optionally search bounded substring	s.index(<substring>)	Index of first match or raise ValueError
s.endswith(<suffix>[, start[, end]])	True if string ends with suffix, optionally search bounded substring	s.lower()	To lower case
s.strip(chars=None)	Strip whitespace from both ends, or passed characters	s.upper()	To upper case
s.lstrip(chars=None)	Strip whitespace from left end, or passed characters	s.title()	To title case (The Quick Brown Fox)
s.rstrip(chars=None)	Strip whitespace from right end, or passed characters	s.capitalize()	Capitalize first letter
s.ljust(width, fillchar=' ')	Left justify with fillchar	s.replace(old, new[, count])	Replace <i>old</i> with <i>new</i> at most <i>count</i> times
s.rjust(width, fillchar=' ')	Right justify with fillchar	s.translate(<table>)	Use <i>str.maketrans(&lt;dict&gt;)</i> to generate table
s.center(width, fillchar=' ')	Center with fillchar	chr(<int>)	Integer to Unicode character
s.split(sep=None, maxsplit=-1)	Split on whitespace, or <i>sep</i> str at most <i>maxsplit</i> times	ord(<str>)	Unicode character to integer
s.splitlines(keepends=False)	Split lines on <code>[\n\r\f\v\x1c-\x1e\x85\u2028\u2029]</code> and <code>\r\n</code>	s.isdecimal()	True if <code>[0-9]</code> , <code>[o-9]</code> or <code>[9-0]</code>
<separator>.join(<strings>)	Join sequence of <i>strings</i> with <i>separator</i> string	s.isdigit()	True if <code>isdecimal()</code> or <code>[231...]</code>
		s.isnumeric()	True if <code>isdigit()</code> or <code>[¼¾零〇一...]</code>
		s.isalnum()	True if <code>isnumeric()</code> or <code>[a-zA-Z...]</code>
		s.isprintable()	True if <code>isalnum()</code> or <code>[!#\$%...]</code>
		s.isspace()	True if <code>[\t\n\r\f\v\x1c-\x1f\x85\xa0...]</code>

```
head, sep, tail = s.partition(<separator>)
# Search for separator from start and split
```

```
head, sep, tail = s.rpartition(<separator>)
# Search for separator from end and split
```

```
s.removeprefix(<prefix>)
# Remove prefix if present (3.9+)
```

```
s.removesuffix(<suffix>)
# Remove suffix if present (3.9+)
```

### String formatting

f-string	Output
<code>f"{6/3}, {'a'+'b'}"</code> <code>'{ }, { }'.format(6/3, 'a'+'b')</code>	<code>'2, ab'</code>
<code>f'{1:&lt;5}'</code>	<code>'1      '</code>
<code>f'{1:^5}'</code>	<code>'      1      '</code>
<code>f'{1:&gt;5}'</code>	<code>'                1'</code>
<code>f'{1:.&lt;5}'</code>	<code>'1....'</code>
<code>f'{1:.&gt;5}'</code>	<code>'....1'</code>
<code>f'{1:0}'</code>	<code>'1'</code>
<code>f'{1+1=}'</code>	<code>'1+1=2' (= prepends)</code>
<code>f'{v!r}'</code>	<code>repr(v)</code>
<code>f'{today:%d %b %Y}'</code>	<code>'21 Jan 1984'</code>
<code>f'{1.729:.2f}'</code>	<code>'1.73'</code>
<code>f'{1.7:04}'</code>	<code>'01.7'</code>
<code>f'{1.7:4}'</code>	<code>' 1.7'</code>
<code>f"{'abc':.2}"</code>	<code>'ab'</code>
<code>f"{'abc':6.2}"</code>	<code>'ab      '</code>
<code>f"{'abc'!r:6}"</code>	<code>""'abc' ""</code>
<code>f'{123456:,}'</code>	<code>'123,456'</code>
<code>f'{123456:_}'</code>	<code>'123_456'</code>
<code>f'{123456:+6}'</code>	<code>'          +123'</code>
<code>f'{123456:=+6}'</code>	<code>'+   123'</code>
<code>f'{1.234:.2}'</code>	<code>'1.2'</code>
<code>f'{1.234:.2f}'</code>	<code>'1.23'</code>
<code>f'{1.234:.2e}'</code>	<code>'1.230e+00'</code>
<code>f'{1.234:.2%}'</code>	<code>'123.40%'</code>
<code>f'{164:b}'</code>	<code>'10100100'</code>
<code>f'{164:o}'</code>	<code>'244'</code>
<code>f'{164:X}'</code>	<code>'A4'</code>
<code>f'{164:c}'</code>	<code>'ÿ'</code>
<code>f'{1 #comment}'</code>	<code>'1' (v3.12)</code>

### Regex

Standard library `re` module provides Python regular expressions.

```
>>> import re
>>> my_re = re.compile(r'name is (?P<name>[A-Za-z]+)')
>>> match = my_re.search('My name is Douglas.')
>>> match.group()
'name is Douglas'
>>> match.group(1)
'Douglas'
>>> match.groupdict()['name']
'Douglas'
```

### Regex syntax

.	Any character (newline if DOTALL)		Or
^	Start of string (every line if MULTILINE)	(...)	Group
\$	End of string (every line if MULTILINE)	(?:...)	Non-capturing group
*	0 or more of preceding	(?P<name>...)	Named group
+	1 or more of preceding	(?P=name)	Match text matched by earlier group
?	0 or 1 of preceding	(?=...)	Match next, non-consumptive
*?, +?, ??	Same as *, + and ?, as few as possible	(?!...)	Non-match next, non-consumptive
{m,n}	m to n repetitions	(?<=...)	Match preceding, positive lookbehind assertion
{m,n}?	m to n repetitions, as few as possible	(?<!...)	Non-match preceding, negative lookbehind assertion
[...]	Character set: e.g. '[a-zA-Z]'	(?(group)A B)	Conditional match - A if group previously matched else B
[^...]	NOT character set	(?letters)	Set flags for RE ('i', 'L', 'm', 's', 'u', 'x')
\	Escape chars '*?+&\$ ()'', introduce special sequences	(?#...)	Comment (ignored)
\\	Literal '\'		

### Regex special sequences

\<n>	Match by integer group reference starting from 1	\s	Whitespace [ \t\n\r\f\v] (see flag: ASCII)
\A	Start of string	\S	Non-whitespace (see flag: ASCII)
\b	Word boundary (see flag: ASCII LOCALE)	\w	Alphanumeric (see flag: ASCII LOCALE)
\B	Not word boundary (see flag: ASCII LOCALE)	\W	Non-alphanumeric (see flag: ASCII LOCALE)
\d	Decimal digit (see flag: ASCII)	\Z	End of string
\D	Non-decimal digit (see flag: ASCII)		

### Regex flags

Flags modify regex behaviour. Pass to regex functions `re.A` / `re.ASCII` or embed in regular expression `(?a)`.

(?a)   A   ASCII	ASCII-only match for \w, \W, \b, \B, \d, \D, \s, \S (default is Unicode)	(?m)   M   MULTILINE	Match every new line, not only start/end of string
(?i)   I   IGNORECASE	Case insensitive matching	(?s)   S   DOTALL	'.' matches ALL chars, including newline
(?L)   L   LOCALE	Apply current locale for \w, \W, \b, \B (discouraged)	(?x)   X   VERBOSE	Ignores whitespace outside character sets
		DEBUG	Display expression debug info

**Regex functions**

compile(pattern[, flags=0])	Compiles <u>Regular Expression Obj</u>	findall(pattern, string)	Non-overlapping matches as list of groups or tuples (>1)
escape(string)	Escape non-alphanumerics	finditer(pattern, string[, flags])	Iterator over non-overlapping matches
match(pattern, string[, flags])	Match from start	sub(pattern, repl, string[, count=0])	Replace count first leftmost non-overlapping; If repl is function, called with a MatchObj
search(pattern, string[, flags])	Match anywhere	subn(pattern, repl, string[, count=0])	Like sub(), but returns (newString, numberOfSubsMade)
split(pattern, string[, maxsplit=0])	Splits by pattern, keeping splitter if grouped		

**Regex object**

flags	<u>Flags</u>	split(string[, maxsplit=0])	See split() function
groupindex	{group name: group number}	findall(string[, pos[, endpos]])	See findall() function
pattern	Pattern	finditer(string[, pos[, endpos]])	See finditer() function
match(string[, pos][, endpos])	Match from start of target[pos:endpos]	sub(repl, string[, count=0])	See sub() function
search(string[, pos][, endpos])	Match anywhere in target[pos:endpos]	subn(repl, string[, count=0])	See subn() function

**Regex match object**

pos	pos passed to search or match	re	RE object
endpos	endpos passed to search or match		

<code>group([g1, g2, ...])</code>	One or more groups of match One arg, result is a string Multiple args, result is tuple If <code>gi</code> is 0, returns the entire matching string If <code>1 &lt;= gi &lt;= 99</code> , returns string matching group (None if no such group) May also be a group name Tuple of match groups Non-participating groups are None String if <code>len(tuple)==1</code>	<code>span(group)</code>	<code>(start(group), end(group));</code> (None, None) if group didn't contribute
<code>start(group)</code> <code>, end(group)</code>	Indices of start & end of group match (None if group exists but didn't contribute)	<code>string</code>	String passed to <code>match()</code> or <code>search()</code>

**Number**

<code>bool([object])</code> True, False	Boolean
<code>int([float str bool])</code> 5	Integer
<code>float([int str bool])</code> 5.1, 1.2e-4	Float (inexact, compare with <code>math.isclose(&lt;float&gt;, &lt;float&gt;)</code> )
<code>complex(real=0, imag=0)</code> 3 - 2j, 2.1 + 0.8j	Complex
<code>fractions.Fraction(&lt;numerator&gt;, &lt;denominator&gt;)</code>	Fraction
<code>decimal.Decimal([str int])</code>	Decimal (exact, set precision: <code>decimal.getcontext().prec = &lt;int&gt;</code> )
<code>bin([int])</code> 0b101010 <code>int('101010', 2)</code> <code>int('0b101010', 0)</code>	Binary
<code>hex([int])</code> 0x2a <code>int('2a', 16)</code> <code>int('0x2a', 0)</code>	Hex

**Mathematics**

Also see [built-in](#) functions `abs`, `pow`, `round`, `sum`, `min`, `max`.

```
from math import (e, pi, inf, nan, isinf, isnan,
                  sin, cos, tan, asin, acos, atan, degrees, radians,
                  log, log10, log2)
```

**Statistics**

```
from statistics import mean, median, variance, stdev, quantiles, groupby
```

**Random**

```
>>> from random import random, randint, choice, shuffle, gauss, triangular, seed
>>> random() # float inside [0, 1)
0.42
>>> randint(1, 100) # int inside [<from>, <to>]
42
>>> choice(range(100)) # random item from sequence
42
```

**Time**

The *datetime* module provides immutable hashable *date*, *time*, *datetime*, and *timedelta* classes.

**Time formatting**

Code	Output
%a	Day name short (Mon)
%A	Day name full (Monday)
%b	Month name short (Jan)
%B	Month name full (January)
%c	Locale datetime format
%d	Day of month [01,31]
%f	Microsecond [000000,999999]
%H	Hour (24-hour) [00,23]
%I	Hour (12-hour) [01,12]
%j	Day of year [001,366]
%m	Month [01,12]
%M	Minute [00,59]
%p	Locale format for AM/PM
%S	Second [00,61]. Yes, 61!
%U	Week number (Sunday start) [00(partial),53]
%w	Day number [0(Sunday),6]
%W	Week number (Monday start) [00(partial),53]
%x	Locale date format
%X	Locale time format
%y	Year without century [00,99]
%Y	Year with century (2023)
%Z	Time zone ('' if no TZ)
%z	UTC offset (+HHMM/-HHMM, '' if no TZ)
%%	Literal '%'

**Exception**

```
try:
    ...
except [Exception[ as e]]:
    ...
except: # catch all
    ...
else: # if no exception
    ...
finally: # always executed
    ...

raise exception[ from None] # stop exception chain

try:
    1 / 0
except ZeroDivisionError:
    raise TypeError("Stop chain") from None
```

BaseException	Base class for all exceptions
└ BaseExceptionGroup	Base class for groups of exceptions
└ GeneratorExit	Generator close() raises to terminate iteration
└ KeyboardInterrupt	On user interrupt key (often 'CTRL-C')
└ SystemExit	On sys.exit()
└ Exception	Base class for errors
└ ArithmeticError	Base class for arithmetic errors
└ FloatingPointError	Floating point operation failed
└ OverflowError	Result too large
└ ZeroDivisionError	Argument of division or modulo is 0
└ AssertionError	Assert statement failed
└ AttributeError	Attribute reference or assignment failed
└ BufferError	Buffer operation failed
└ EOFError	input() hit end-of-file without reading data
└ ExceptionGroup	Group of exceptions raised together
└ ImportError	Import statement failed
└ ModuleNotFoundError	Module not able to be found
└ LookupError	Base class for lookup errors
└ IndexError	Index not found in sequence
└ KeyError	Key not found in dictionary
└ MemoryError	Operation ran out of memory
└ NameError	Local or global name not found
└ UnboundLocalError	Local variable value not assigned
└ OSError	System related error
└ BlockingIOError	Non-blocking operation will block
└ ChildProcessError	Operation on child process failed
└ ConnectionError	Base class for connection errors
└ BrokenPipeError	Write to closed pipe or socket
└ ConnectionAbortedError	Connection aborted
└ ConnectionRefusedError	Connection denied by server
└ ConnectionResetError	Connection reset mid-operation
└ FileExistsError	Trying to create a file that already exists
└ FileNotFoundError	File or directory not found
└ InterruptedError	System call interrupted by signal
└ IsADirectoryError	File operation requested on a directory
└ NotADirectoryError	Directory operation requested on a non-directory
└ PermissionError	Operation has insufficient access rights
└ ProcessLookupError	Operation on process that no longer exists
└ TimeoutError	Operation timed out
└ ReferenceError	Weak reference used on garbage collected object
└ RuntimeError	Error detected that doesn't fit other categories
└ NotImplementedError	Operation not yet implemented
└ RecursionError	Maximum recursion depth exceeded
└ StopAsyncIteration	Iterator __anext__() raises to stop iteration
└ StopIteration	Iterator next() raises when no more values
└ SyntaxError	Python syntax error
└ IndentationError	Base class for indentation errors
└ TabError	Inconsistent tabs or spaces
└ SystemError	Recoverable Python interpreter error
└ TypeError	Operation applied to wrong type object
└ ValueError	Operation on right type but wrong value
└ UnicodeError	Unicode encoding/decoding error
└ UnicodeDecodeError	Unicode decoding error
└ UnicodeEncodeError	Unicode encoding error
└ UnicodeTranslateError	Unicode translation error
Warning	Base class for warnings
└ BytesWarning	Warnings about bytes and bytearrays
└ DeprecationWarning	Warnings about deprecated features
└ EncodingWarning	Warning about encoding problem
└ FutureWarning	Warnings about future deprecations for end users
└ ImportWarning	Possible error in module imports
└ PendingDeprecationWarning	Warnings about pending feature deprecations
└ ResourceWarning	Warning about resource use
└ RuntimeWarning	Warning about dubious runtime behavior
└ SyntaxWarning	Warning about dubious syntax
└ UnicodeWarning	Warnings related to Unicode
└ UserWarning	Warnings generated by user code



**Execution**

```
$ python [-bBdEhiIOqsSuvVWx?] [-c command | -m module-name | script | - ] [args]
$ python --version
Python 3.10.12
$ python --help[-all] # help-all [3.11+]
# Execute code from command line
$ python -c 'print("Hello, world!")'
# Execute __main__.py in directory
$ python <directory>
# Execute module as __main__
$ python -m timeit -s 'setup here' 'benchmarked code here'
# Optimise execution
$ python -O script.py

# Hide warnings
PYTHONWARNINGS="ignore"
# OR
$ python -W ignore foo.py
# OR
import warnings
warnings.filterwarnings("ignore", category=DeprecationWarning)
```

```
if __name__ == '__main__': # run main() if file executed as script
    main()
```

**Environment variables**

PYTHONHOME	Change location of standard Python libraries	PYTHONOPTIMIZE	Optimise execution (-O)
PYTHONPATH	Augment default search path for module files	PYTHONWARNINGS	Set warning level [default/error/always/module/once/ignore] (-W)
PYTHONSTARTUP	Module to execute before entering interactive prompt	PYTHONPROFILEIMP ORTTIME	Show module import times (-X)

**sitecustomize.py / usercustomize.py**

Before `__main__` module is executed Python automatically imports:

- sitecustomize.py in the system site-packages directory
- usercustomize.py in the user site-packages directory

```
# Get user site packages directory
$ python -m site --user-site

# Bypass sitecustomize.py/usercustomize.py hooks
$ python -S script.py
```