The Best* Python Cheat Sheet

Just what you need

Keywords

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

*Soft keywords

Operators

Precedence (high->low)	Description
`(…,) […,] {…,} {…:…,}`	tuple, list, set, dict
`s[i] s[i:j] s.attr f(…)`	index, slice, attribute, function call
`await x`	await expression
`+x, -x, ~x`	unary positive, negative, bitwise NOT
`x ** y`	power
`x * y, x @ y, x / y, x // y, x % y`	multiply, maxtrix multply, divide, floor divide, modulus
`x + y, x - y`	add, substract
`x << y  x >> y`	bitwise shift left, right
`x & y`	bitwise and
`x ^ y`	bitwise exclusive or
`x | y`	bitwise or

`x<y  x<=y  x>y  x>=y  x==y x!=y`
`x is y   x is not y`
`x in s   x not in s`
	comparison,
	identity,
	membership

`not x`	boolean negation
`x and y`	boolean and
`x or y`	boolean or
`if - else`	conditional expression
`lambda`	lambda expression
`:=`	assignment expression

Assignment	Usually equivalent
`a = b`	Assign object b to label a
`a += b`	`a = a + b`
`a -= b`	`a = a - b`
`a *= b`	`a = a * b`
`a /= b`	`a = a / b` (true division)
`a //= b`	`a = a // b` (floor division)
`a %= b`	`a = a % b`
`a **= b`	`a = a ** b`
`a &= b`	`a = a & b`
`a |= b`	`a = a | b`

Assignment	Usually equivalent
`a ^= b`	`a = a ^ b`
`a >>= b`	`a = a >> b`
`a <<= b`	`a = a << b`

Splat * operator

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(**kwds): …               # f(x=1, y=2, z=3)
def f(x, **kwds): …            # f(x=1, y=2, z=3) | f(1, y=2, z=3)
def f(*args, **kwds): …        # f(x=1, y=2, z=3) | f(1, y=2, z=3) | f(1, 2, z=3) | f(1, 2,
def f(x, *args, **kwds): …     # f(x=1, y=2, z=3) | f(1, y=2, z=3) | f(1, 2, z=3) | f(1, 2,
def f(*args, y, **kwds): …     # 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
kwds = {'x': 3, 'y': 4}        # ** expands dictionary to keyword arguments
func(*args, **kwds)            # is the same as:
func(1, 2, x=3, y=4)
```

Unpacking

```
head, *body       = s          # unpack assignment
head, *body, tail = s
*body, tail       = s
s  = [*it[, …]]                # unpack to list
s  = (*it[, …])                # unpack to tuple
s  = {*it[, …]}                # 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 expr raise AssertionError(message)
```

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

<expression1> if <condition> else <expression2>

with <expression> [as name]:
  …
```

Match

Python 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

with statements require object with `__enter__()` and `__exit__()` special methods: • `__enter__()` should lock resources and optionally return an object • `__exit()__` should release resources, and handle any exceptions 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!
```

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

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

String

Immutable sequence of characters.

`<substring> in s`	True if string contains *substring*
`s.startswith(<prefix>[, start[, end]])`	True if string starts with prefix, optionally search bounded substring
`s.endswith(<suffix>[, start[, end]])`	True if string ends with suffix, optionally search bounded substring
`s.strip(chars=None)`	Strip whitespace from both ends, or passed characters
`s.lstrip(chars=None)`	Strip whitespace from left end, or passed characters
`s.rstrip(chars=None)`	Strip whitespace from right end, or passed characters
`s.ljust(width, fillchar=' ')`	Left justify with fillchar
`s.rjust(width, fillchar=' ')`	Right justify with fillchar

`s.center(width, fillchar=' ')`	Center with fillchar
`s.rstrip(chars=None)`	Strip whitespace from right end, or passed characters
`s.split(sep=None, maxsplit=-1)`	Split on whitespace, or *sep* str at most *maxsplit* times
`s.splitlines(keepends=False)`	Split lines on [\n\r\f\v\x1c-\x1e\x85\u2028\u2029] and \r\n
`<separator>.join(<strings>)`	Join *strings* with *separator*
`s.find(<substring>)`	Index of first match or -1
`s.index(<substring>)`	Index of first match or raise ValueError
`s.lower()`	To lower case
`s.upper()`	To upper case
`s.title()`	To title case (The Quick Brown Fox)
`s.capitalize()`	Capitalize first letter
`s.replace(old, new[, count])`	Replace *old* with *new* at most *count* times
`s.translate(<table>)`	Use *str.maketrans(<dict>)* to generate table
`chr(<int>)`	Integer to Unicode character
`ord(<str>)`	Unicode character to integer
`s.isdecimal()`	True if [0-9], [०-९] or [٩-٠]
`s.isdigit()`	True if isdecimal() or [²³¹…]
`s.isnumeric()`	True if isdigit() or [¼½¾零〇一…]
`s.isalnum()`	True if isnumeric() or [a-zA-Z…]
`s.isprintable()`	True if isalnum() or [!
`s.isspace()`	True if [\t\n\r\f\v\x1c-\x1f\x85\xa0…]
`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

String formatting

f-string	Output

`f"{6/3}, {'a'+'b'}"`
`'{}, {}'.format(6/3, 'a'+'b')`
	`'2, ab'`

`f'{1:<5}'`	`'1    '`
`f'{1:^5}'`	`'       1  '`
`f'{1:>5}'`	`'         1'`
`f'{1:.<5}'`	`'1....'`
`f'{1:.>5}'`	`'....1'`
`f'{1:0}'`	`'1'`
`f'{1+1=}'`	`'1+1=2'` (= prepends)
`f'{v!r}'`	`repr(v)`
`f'{today:%d %b %Y}'`	`'21 Jan 1984'`
`f'{1.729:.2f}'`	`'1.73'`
`f'{1.7:04}'`	`'01.7'`
`f'{1.7:4}'`	`' 1.7'`
`f"{'abc':.2}"`	`'ab'`
`f"{'abc':6.2}"`	`'ab    '`
`f"{'abc'!r:6}"`	`"'abc' "`
`f'{123456:,}'`	`'123,456'`
`f'{123456:_}'`	`'123_456'`
`f'{123456:+6}'`	`'       +123'`

`f'{123456:=+6}'`	`'+  123'`
`f'{1.234:.2}'`	`'1.2'`
`f'{1.234:.2f}'`	`'1.23'`
`f'{1.234:.2e}'`	`'1.230e+00'`
`f'{1.234:.2%}'`	`'123.40%'`
`f'{164:b}'`	`'10100100'`
`f'{164:o}'`	`'244'`
`f'{164:X}'`	`'A4'`
`f'{164:c}'`	`'ÿ'`
`f'{1 #comment}'`	`'1' (v3.12)`

Regex

```
>>> 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)
^	Start of string (every line if MULTILINE)
$	End of string (every line if MULTILINE)
*	0 or more of preceding
+	1 or more of preceding
?	0 or 1 of preceding
*?, +?, ??	Same as *, + and ?, as few as possible
{m,n}	m to n repetitions
{m,n}?	m to n repetitions, as few as possible
[ ]	Character set: e.g. '[a-zA-Z]'
[^]	NOT character set
\	Escape chars '*?+&$|()', introduce special sequences

\\	Literal '\' Or
(…)	Group
(?:…)	Non-capturing group
(?P<name>…)	Named group
(?P=name)	Match text matched by earlier group
(?=…)	Match next, non-consumptive
(?!…)	Non-match next, non-consumptive
(?<=…)	Match preceding, positive lookbehind assertion
(?<!…)	Non-match preceding, negative lookbehind assertion
(?(group)A|B)	Conditional match - A if group previously matched else B
(?letters)	Set flags for RE ('i','L', 'm', 's', 'u', 'x')
(?#…)	Comment (ignored)

Regex special sequences

\number	Match by group reference starting from 1
\A	Start of string
\b	Word boundary
\B	Not word boundary
\d	Decimal digit
\D	Non-decimal digit

\s	Whitespace [\t\n\r\f\v]
\S	Non-whitespace
\w	Alphanumeric (depends on LOCALE flag)
\W	Non-alphanumeric
\Z	End of string

Regex flags

I or IGNORECASE <=> (?i)	Case insensitive matching
L or LOCALE <=> (?L)	\w, \W, \b, \B depend on current locale

M or MULTILINE <=> (?m)	Match every new line, not only start/end of string
S or DOTALL <=> (?s)	

	'.' matches ALL chars, including newline
U or UNICODE <=> (?u)	\w, \W, \b, and \B dependent on Unicode da

	tabase
X or VERBOSE <=> (?x)	Ignores whitespace outside character sets

Regex functions

compile(pattern[,flags=0])	Compiles *Regular Expression Object*
escape(string)	Escape non-alphanumerics
match(pattern, string[, flags])	Match from start
search(pattern, string[, flags])	Match anywhere
split(pattern, string[, maxsplit=0])	Splits by pattern, keeping splitter if grouped

findall(pattern, string)	Non-overlapping matches as list of groups or tuples (>1)
finditer(pattern, string[, flags])	Iterator over non-overlapping matches
sub(pattern, repl, string[, count=0])	Replace count first leftmost non-overlapping; If repl is function, called with a MatchObj
subn(pattern, repl, string[, count=0])	Like sub(), but returns (newString, numberOfSubsMade)

Regex objects

flags	Flags
groupindex	{group name: group number}
pattern	Pattern
match(string[, pos][, endpos])	Match from start of target[pos:endpos]
search(string[, pos][, endpos])	Match anywhere in target[pos:endpos]

split(string[, maxsplit=0])	See split() function
findall(string[, pos[, endpos]])	See findall() function
finditer(string[, pos[, endpos]])	See finditer() function
sub(repl, string[, count=0])	See sub() function
subn(repl, string[, count=0])	See subn() function

Regex match objects

pos	pos passed to search or match
endpos	endpos passed to search or match
re	RE object
group([g1, g2, ...])	One or more groups of match One arg, result is a string Multiple args, result is tuple If gi is 0, returns the entire matching string If 1 <= gi <= 99, 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 len(tuple)==1
start(group), end(group)	Indices of start & end of group match (None if group exists but didn't contribute)
span(group)	(start(group), end(group)); (None, None) if group didn't contibute
string	String passed to match() or search()

Math / Numbers

int(<float|str|bool>) 5	Integer
float(<int|str|bool>)	Float (inexact, compare with math.isclose(<float>, <float>)
5.1, 1.2e-4	
complex(real=0, imag=0) 3 - 2j, 2.1 + 0.8j	Complex
fractions.Fraction(<numerator>, <denominator>)	Fraction

decimal.Decimal(<str|int>)	Decimal (exact, set precision: decimal.getcontext().prec = <int>)

bin(<int>)
0b101010
int('101010', 2)
int('0b101010', 0)
	Binary

hex(<int>)
0x2a
int('2a', 16)
int('0x2a', 0)
	Hex

Functions

pow(<num>, <num>) <num> ** <num>	Power	round(<num>[, ±ndigit s])	Round
abs(<num>)	Absolute		

Mathematics

```
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
```

Sequence

Operations on sequence types (List, Tuple, String)

x in s	True if any s[i]==x	max(s)	Largest item
x not in s	True if no s[i]==x	s.index(x[, start[, stop]])	Smallest i where s[i]==x, start/stop bounds search
s1 + s2	Concatenate s1 and s2	reversed(s)	Iterator on s in reverse order (use reversed(list(<str>)))
s*n, n*s	Concatenate n copies of s		
s.count(x)	Count of s[i]==x		
len(s)	Number of items	sorted(s1, cmp=func, key=getter, reverse=False)	New sorted list
min(s)	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
[1, 4]
>>> s[:2]               # slices are created implicitly when indexing
[0, 1]
>>> s[3::-1]            # negative steps
[3, 2, 1, 0]
>>> s[1:3]
[1, 2]
>>> s[1:5:2]
[1, 3]
```

Tuple

Immutable hashable sequence.

s = (1, 'a', 3.0)
s = 1, 'a', 3.0
	Create tuple

s = (1,)	Create single-item tuple
s = ()	Empty tuple
(1, 2, 3) == (1, 2) + (3,)	Add makes new tuple
(1, 2, 1, 2) == (1, 2) * 2	Multply makes new tuple

Named tuple

Subclass with named items.

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

List

Mutable sequence (non-hashable).

s = [1, 'a', 3.0]
s = list(range(3))
	Create list

s[i] = x	Replace item index i with x
s[<slice>] = it	Replace slice with iterable
del s[<slice>] s[<slice>] = []	Delete slice

s.append(x)
s += x
s[len(s):len(s)] = [x]
	Add element to end

s.extend(it) s[len(s):len(s)] = it	Add elements from iterable to end
s.insert(i, x) s[i:i] = [x]	Insert item at index i
s.remove(x) del s[s.index(x)]	Remove item
y = s.pop([i])	Remove and return last, or by index
s.reverse()	Reverse in place
s.sort(cmp=func, key=getter, reverse=False)	Sort in place, default ascending

Dictionary

Mutable key:value pair mapping (non-hashable).

dict()
{}
	Empty dict

dict(<sequence|mapping>)	Create from key:value pairs
dict(**kwds)	Create from keyword arguments
dict(zip(keys, values))	Create from sequences of keys and values
dict.fromkeys(keys, value=None)	Create from keys, all set to value
d.keys()	Iterable of keys
d.values()	Iterable of values
d.items()	Iterable of (key, value) pairs
d.get(key, default=None)	Get value for key, or default
d.setdefault(key, default=None)	Get value for key, add if missing
d.pop(key)	Remove and return value for key, raise KeyError if missing
d.popitem()	Remove and return (key, value) pair (last-in, first-out)
d.clear()	Remove all items
d.copy()	Shallow copy

collections.defaultdict(<type>)
collections.defaultdict(lambda: 42)
	dict with default value <type>()
	e.g. dict with default value 42

d1.update(d2) d1 |= d2	Add/replace key:value pairs from d2 to d1 |= syntax Python 3.9+

d3 = d1 | d2
d3 = {**d1, **d2}
	Merge to new dict, d2 trumps d1

{k for k, v in d.items() if v==value}	Set of keys with given value

Set

Mutable (set) and immutable (frozenset) sets.

set(iterable=None)
{1, 2, 3}
frozenset(iterable=None)
	New set from iterable, or empty
	But {} creates an empty dictionary (sad!)

len(s)	Cardinality
v in s v not in s	Test membership
s1.issubset(s2)	True if s1 is subset of s2
s1.issuperset(s2)	True if s1 is superset of s2
s.add(v)	Add element
s.remove(v)	Remove element (KeyError if not found)
s.discard(v)	Remove element if present
s.pop()	Remove and return arbitrary element (KeyError if empty)
s.clear()	Remove all elements
s1.intersection(s2[, s3…]) s1 & s2	New set of shared elements
s1.union(s2[, s3…]) s1 | s2	New set of all elements
s1.difference(s2[, s3…]) s1 - s2	New set of elements unique to s1
s1.symmetric_difference(s2) s1 ^ s2	New set of unshared elements
s.copy()	Shallow copy
s.update(it1[, it2…])	Add all values from iterables

Bytes

Immutable sequence of bytes. Mutable version is bytearray.

b'<str>'	Create bytes, from ASCII characters and x00-xff
bytes(<ints>)	Create from int sequence
bytes(<str>, 'utf-8') <str>.encode('utf-8')	Create from string
<int>.to_bytes(length, order, signed=False)	Create from int (order='big'|'little')
bytes.fromhex('<hex>')	Create from hex pairs (can be separated by whitespace)
<separator>.join(<byte_objs>)	Join ^byte_objs^ with ^separator^
list(<bytes>)	Returns ints in range from 0 to 255
str(<bytes>, 'utf-8') <bytes>.decode('utf-8')	
int.from_bytes(bytes, order, signed=False)	Return int from bytes (order='big'|'little')
<bytes>.hex(sep='', bytes_per_sep=2)	Return hex pairs

```python
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)
```

Built-in functions

abs()	The absolute value of a number
aiter()	Asynchronous iterator for an asynchronous iterable
all()	True if all elements of iterable are true
any()	True if any element of iterable is true
ascii()	A string with a printable representation of an object
bin()	Convert an integer number to a binary string
bool()	Boolean value
breakpoint()	Drop into debugger at call site
bytearray()	New array of bytes
bytes()	New “bytes” object
callable()	True if the object argument is callable

`chr()`	The string representing a character
`classmethod()`	Transform method into class method
`compile()`	Compile source into code or AST object
`complex()`	Complex number with the value real + imag*1j
`delattr()`	Delete the named attribute, if object allows
`dict()`	Create new dictionary
`dir()`	List of names in the current local scope
`divmod()`	Pair of numbers (quotient, remainder)
`enumerate()`	Enumerate object
`eval()`	Execute expression
`exec()`	Execution Python code
`filter()`	Make iterator from an iterable and returns true
`float()`	Floating point number from number or string
`format()`	Formatted representation
`frozenset()`	New frozenset object.
`getattr()`	Get value of named attribute of object
`globals()`	Dictionary of current module namespace
`hasattr()`	True if object has named attribute
`hash()`	Hash value of object
`help()`	Built-in help system
`hex()`	Convert integer to lowercase hexadecimal string
`id()`	Return unique integer identifier of object
`__import__()`	Invoked by the import statement
`input()`	input and converts it into a string
`int()`	Create integer from number or string
`isinstance()`	True if object is instance of a class object
`issubclass()`	True if class is a subclass of class object
`iter()`	Iterator for object
`len()`	Length of an object
`list()`	Create list
`locals()`	Dictionary of current local symbol table
`map()`	Apply function to every item of iterable
`max()`	Largest item in an iterable
`memoryview()`	Access internal object data via buffer protocol
`min()`	Smallest item in an iterable
`next()`	Next item from iterator
`object()`	New featureless object
`oct()`	Convert integer to octal string
`open()`	Open file object
`ord()`	Integer representing Unicode code point of character
`pow()`	Return base to the power exp.
`print()`	Print object to text stream file
`property()`	Property decorator
`range()`	Generate integer sequences
`repr()`	String representation of object for debugging
`reversed()`	Reverse iterator
`round()`	Number rounded to ndigits precision after decimal point
`set()`	New set object.
`setattr()`	Set object attribute value by name
`slice()`	Slice object representing a set of indices
`sorted()`	New sorted list from the items in iterable
`staticmethod()`	Transform method into static method
`str()`	String description of object
`sum()`	Sums items of iterable
`super()`	Proxy object that delegates method calls to parent or sibling
`tuple()`	Create a tuple
`type()`	Type of an object
`vars()`	dict attribute for any other object with a dict attribute
`zip()`	Iterate over multiple iterables in parallel

Time

Provides 'date', 'time', 'datetime' and 'timedelta' classes. All are immutable and hashable.

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 '%'

Exceptions

```
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 asssigned
   ├─ 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 insuffient 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 bytesarrays
      ├─ 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 / Environment

```
$ python [-bBdEhiIOqsSuvVWx?] [-c command | -m module-name | script | - ] [args]
$ python --help[-all] # help-all Python 3.11+
# Execute code from command line
$ python -c 'print("Hello, world!")'
# 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	PYTHONPROFILEIMPORTTIME	Show module import times (-X)

sitecustomize.py / usercustomize.py

Before `__main__` 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
```