

The Best* Python Cheat Sheet

Just what you need

Keywords

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

*Soft keywords

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	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 (<i>closure</i>) By default, assignment creates a new local name <i>global <name></i> grants read/write access to specified module name <i>nonlocal <name></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
```

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 multiply, divide, floor divide, modulus
x + y, x - y	add, subtract
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	comparison,
x is y x is not y	identity,
x in s x not in s	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
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(**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)

```

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**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>[, ...[, **kws]]}	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!

```

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, **kwargs)
if isinstance(obj, cls):
    obj.__init__(*args, **kwargs)

```

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>
divmod(self, other)	<code>--divmod__(self, other)</code>
divmod(self, other)	<code>--rdivmod__(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>

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 < other</code>	<code>__lt__(self, other)</code>
<code>self <= other</code>	<code>__le__(self, other)</code>
<code>self > other</code>	<code>__gt__(self, other)</code>
<code>self >= 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, **kws)</code>	<code>__call__(self, *args, **kws)</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>__getattr__(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>

String

Immutable sequence of characters.

The Best* Python Cheat Sheet

<code><substring> in s</code>	True if string contains <i>substring</i>	<code>s.index(<substring>)</code>	Index of first match or raise <code>ValueError</code>
<code>s.startswith(<prefix>[, start[, end]])</code>	True if string starts with prefix, optionally search bounded substring	<code>s.lower()</code>	To lower case
<code>s.endswith(<suffix>[, start[, end]])</code>	True if string ends with suffix, optionally search bounded substring	<code>s.upper()</code>	To upper case
<code>s.strip(chars=None)</code>	Strip whitespace from both ends, or passed characters	<code>s.title()</code>	To title case (The Quick Brown Fox)
<code>s.lstrip(chars=None)</code>	Strip whitespace from left end, or passed characters	<code>s.capitalize()</code>	Capitalize first letter
<code>s.rstrip(chars=None)</code>	Strip whitespace from right end, or passed characters	<code>s.replace(old, new[, count])</code>	Replace <i>old</i> with <i>new</i> at most <i>count</i> times
<code>s.ljust(width, fillchar=' ')</code>	Left justify with fillchar	<code>s.translate(<table>)</code>	Use <i>str.maketrans(<dict>)</i> to generate table
<code>s.rjust(width, fillchar=' ')</code>	Right justify with fillchar	<code>chr(<int>)</code>	Integer to Unicode character
<code>s.center(width, fillchar=' ')</code>	Center with fillchar	<code>ord(<str>)</code>	Unicode character to integer
<code>s.rstrip(chars=None)</code>	Strip whitespace from right end, or passed characters	<code>s.isdecimal()</code>	True if <code>[0-9]</code> , <code>[o-9]</code> or <code>[١-٩]</code>
<code>s.split(sep=None, maxsplit=-1)</code>	Split on whitespace, or <i>sep</i> str at most <i>maxsplit</i> times	<code>s.isdigit()</code>	True if <code>isdecimal()</code> or <code>[٢٣١...]</code>
<code>s.splitlines(keepend s=False)</code>	Split lines on <code>[\n\r\f\v\x1c-\x1e\x85\u2028\u2029]</code> and <code>\r\n</code>	<code>s.isnumeric()</code>	True if <code>isdigit()</code> or <code>[¼½¾零〇一...]</code>
<code><separator>.join(<strings>)</code>	Join <i>strings</i> with <i>separator</i>	<code>s.isalnum()</code>	True if <code>isnumeric()</code> or <code>[a-zA-Z...]</code>
<code>s.find(<substring>)</code>	Index of first match or -1	<code>s.isprintable()</code>	True if <code>isalnum()</code> or <code>[!</code>
		<code>s.isspace()</code>	True if <code>[\t\n\r\f\v\x1c-\x1f\x85\xa0...]</code>
		<code>head, sep, tail = s.partition(<separator>)</code>	Search for separator from start and split
		<code>head, sep, tail = s.rpartition(<separator>)</code>	Search for separator from end and split

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:<5}'</code>	<code>'1 '</code>
<code>f'{1:^5}'</code>	<code>' 1 '</code>
<code>f'{1:>5}'</code>	<code>' 1'</code>
<code>f'{1:.<5}'</code>	<code>'1....'</code>
<code>f'{1:.>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]
\A	Start of string	\S	Non-whitespace
\b	Word boundary	\w	Alphanumeric (depends on LOCALE flag)
\B	Not word boundary	\W	Non-alphanumeric
\d	Decimal digit	\Z	End of string
\D	Non-decimal digit		

Regex flags

I or IGNORECASE <=> (?i)	Case insensitive matching	S or DOTALL <=> (?s)	'.' matches ALL chars, including newline
L or LOCALE <=> (?L)	\w, \W, \b, \B depend on current locale	U or UNICODE <=> (?u)	\w, \W, \b, and \B dependent on Unicode database
M or MULTILINE <=> (?m)	Match every new line, not only start/end of string	X or VERBOSE <=> (?x)	Ignores whitespace outside character sets

Regex functions

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

<code>findall(pattern, string)</code>	Non-overlapping matches as list of groups or tuples (>1)	<code>sub(pattern, repl, string[, count=0])</code>	Replace count first leftmost non-overlapping; If repl is function, called with a MatchObj
<code>finditer(pattern, string[, flags])</code>	Iterator over non-overlapping matches	<code>subn(pattern, repl, string[, count=0])</code>	Like sub(), but returns (newString, numberOfSubsMade)
Regex objects			
<code>flags</code>	Flags	<code>split(string[, maxsplit=0])</code>	See split() function
<code>groupindex</code>	{group name: group number}	<code>findall(string[, pos[, endpos]])</code>	See findall() function
<code>pattern</code>	Pattern	<code>finditer(string[, pos[, endpos]])</code>	See finditer() function
<code>match(string[, pos[, endpos]])</code>	Match from start of target[pos:endpos]	<code>sub(repl, string[, count=0])</code>	See sub() function
<code>search(string[, pos[, endpos]])</code>	Match anywhere in target[pos:endpos]	<code>subn(repl, string[, count=0])</code>	See subn() function
Regex match objects			
<code>pos</code>	pos passed to search or match	<code>start(group)</code>	Indices of start & end of , end(group)
<code>endpos</code>	endpos passed to search or match	<code>end(group)</code>	Indices of start & end of group match (None if group exists but didn't contribute)
<code>re</code>	RE object	<code>span(group)</code>	(start(group), end(group)); (None, None) if group didn't contribute
<code>group([g1, g2, ...])</code>	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	<code>string</code>	String passed to match() or search()

Numbers / Math

<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(<float>, <float>)</code>)
<code>complex(real=0, imag=0)</code> 3 - 2j, 2.1 + 0.8j	Complex
<code>fractions.Fraction(<numerator>, <denominator>)</code>	Fraction

<code>decimal.Decimal(<str int>)</code>	Decimal (exact, set precision: <code>decimal.getcontext().prec = <int></code>)
<code>bin(<int>)</code> <code>0b101010</code> <code>int('101010', 2)</code> <code>int('0b101010', 0)</code>	Binary
<code>hex(<int>)</code> <code>0x2a</code> <code>int('2a', 16)</code> <code>int('0x2a', 0)</code>	Hex
Functions	
<code>pow(<num>, <num>)</code> <code><num> ** <num></code>	Power
<code>abs(<num>)</code>	Absolute
<code>round(<num>[, ±ndigits])</code>	Round

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).

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

```

- 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> <code>s = 1, 'a', 3.0</code>	Create tuple
<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.

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

List comprehension

```
result = [expression for item1 in sequence1 {if condition1}
           {for item2 in sequence2 {if condition2} ... for itemN in sequenceN {if conditionN}}]
```

is equivalent to:

```
result = []
for item1 in sequence1:
    for item2 in sequence2:
        ...
        for itemN in sequenceN:
            if condition1 and condition2 ... and conditionN:
                result.append(expression)
```

Dictionary

Mutable non-hashable key:value pair mapping.

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

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

Set

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

<code>set(iterable=None)</code> <code>{1, 2, 3}</code> <code>frozenset(iterable=None)</code>	New set from iterable, or empty But {} creates an empty dictionary (sad!)	<code>s.pop()</code>	Remove and return arbitrary element (<code>KeyError</code> if empty)
<code>len(s)</code>	Cardinality	<code>s.clear()</code>	Remove all elements
<code>v in s</code> <code>v not in s</code>	Test membership	<code>s1.intersection(s2[, s3...])</code> <code>s1 & s2</code>	New set of shared elements
<code>s1.issubset(s2)</code>	True if s1 is subset of s2	<code>s1.union(s2[, s3...])</code> <code>s1 s2</code>	New set of all elements
<code>s1.issuperset(s2)</code>	True if s1 is superset of s2	<code>s1.difference(s2[, s3...])</code> <code>s1 - s2</code>	New set of elements unique to s1
<code>s.add(v)</code>	Add element	<code>s1.symmetric_difference(s2)</code> <code>s1 ^ s2</code>	New set of unshared elements
<code>s.remove(v)</code>	Remove element (<code>KeyError</code> if not found)	<code>s.copy()</code>	Shallow copy
<code>s.discard(v)</code>	Remove element if present	<code>s.update(it1[, it2...])</code>	Add all values from iterables

Bytes

Immutable sequence of bytes. Mutable version is *bytearray*.

<code>b'<str>'</code>	Create from ASCII characters and <code>\x00-\xff</code>	<code>bytes.fromhex('<hex>')</code>	Create from hex pairs (can be separated by whitespace)
<code>bytes(<ints>)</code>	Create from int sequence	<code><int> = <bytes>[<index>]</code>	Return int in range 0 to 255
<code>bytes(<str>, 'utf-8')</code> <code><str>.encode('utf-8')</code>	Create from string	<code><bytes> = <bytes>[<slice>]</code>	Return bytes even if only one element
<code><int>.to_bytes(1, order, signed=False)</code>	Create from int (order='big' 'little')	<code>list(<bytes>)</code>	Return ints in range 0 to 255

<code><bytes_sep>.join(<byte_objs>)</code>	Join <i>byte_objs</i> sequence with <i>bytes_sep</i> separator
<code>str(<bytes>, 'utf-8')</code>	Convert bytes to string
<code><bytes>.decode('utf-8')</code>	

<code>int.from_bytes(b ytes, order, signed=False)</code>	Return int from bytes (order='big' 'little')
<code><bytes>.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)
```

Built-in functions

<code>abs()</code>	Absolute value of number	<code>dir()</code>	List of names in the local scope
<code>aiter()</code>	Asynchronous iterator for an asynchronous iterable	<code>divmod()</code>	Pair of numbers (quotient, remainder)
<code>all()</code>	True if all elements of iterable are true	<code>enumerate()</code>	Enumerate object as (n, item) pairs
<code>any()</code>	True if any element of iterable is true	<code>eval()</code>	Execute expression
<code>ascii()</code>	A string with a printable representation of an object	<code>exec()</code>	Execute Python code
<code>bin()</code>	Convert integer number to binary string	<code>filter()</code>	Make iterator from an iterable, return True
<code>bool()</code>	Boolean value	<code>float()</code>	Floating point number from number or string
<code>breakpoint()</code>	Drop into debugger at call site	<code>format()</code>	Formatted representation
<code>bytearray()</code>	New array of bytes	<code>frozenset()</code>	New frozenset object
<code>bytes()</code>	New bytes object	<code>getattr()</code>	Get value of named attribute of object
<code>callable()</code>	True if the argument is callable	<code>globals()</code>	Dictionary of current module namespace
<code>chr()</code>	One character string for unicode ordinal i (0 <= i <= 0x10ffff)	<code>hasattr()</code>	True if object has named attribute
<code>classmethod()</code>	Transform method into class method	<code>hash()</code>	Hash value of object
<code>compile()</code>	Compile source into code or AST object	<code>help()</code>	Built-in help system
<code>complex()</code>	Complex number with the value real + imag*1j	<code>hex()</code>	Convert integer to lowercase hexadecimal string
<code>delattr()</code>	Delete the named attribute, if object allows	<code>id()</code>	Return unique integer identifier of object
<code>dict()</code>	Create new dictionary	<code>__import__()</code>	Invoked by the import statement
		<code>input(prompt='')</code>	Read string from stdin, with optional prompt

<code>int()</code>	Create integer from number or string	<code>property()</code>	Property decorator
<code>isinstance()</code>	True if object is instance of given class	<code>range()</code>	Generate integer sequence
<code>issubclass()</code>	True if class is subclass of given class	<code>repr()</code>	String representation of object for debugging
<code>iter()</code>	Iterator for object	<code>reversed()</code>	Reverse iterator
<code>len()</code>	Length of object	<code>round()</code>	Number rounded to ndigits precision after decimal point
<code>list()</code>	Create list	<code>set()</code>	New set object
<code>locals()</code>	Dictionary of current local symbol table	<code>setattr()</code>	Set object attribute value by name
<code>map()</code>	Apply function to every item of iterable	<code>slice()</code>	Slice object representing a set of indices
<code>max()</code>	Largest item in an iterable	<code>sorted()</code>	New sorted list from the items in iterable
<code>memoryview()</code>	Access internal object data via buffer protocol	<code>staticmethod()</code>	Transform method into static method
<code>min()</code>	Smallest item in an iterable	<code>str()</code>	String description of object
<code>next()</code>	Next item from iterator	<code>sum()</code>	Sums items of iterable
<code>object()</code>	New featureless object	<code>super()</code>	Proxy object that delegates method calls to parent or sibling
<code>oct()</code>	Convert integer to octal string	<code>tuple()</code>	Create a tuple
<code>open()</code>	Open file object	<code>type()</code>	Type of an object
<code>ord()</code>	Integer representing Unicode code point of character	<code>vars()</code>	dict attribute for any other object with a dict attribute
<code>pow()</code>	Return base to the power exp.	<code>zip()</code>	Iterate over multiple iterables in parallel
<code>print()</code>	Print object to text stream file		

Time

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

Time formatting

Code	Output
<code>%a</code>	Day name short (Mon)
<code>%A</code>	Day name full (Monday)
<code>%b</code>	Month name short (Jan)
<code>%B</code>	Month name full (January)
<code>%c</code>	Locale datetime format
<code>%d</code>	Day of month [01,31]
<code>%f</code>	Microsecond [000000,999999]
<code>%H</code>	Hour (24-hour) [00,23]

Code	Output
%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 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 / Environ

```

$ 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

```