gto76 / python-cheatsheet

Code | Issues 5 | Pull requests 18 | Discussions | Actions | ···

main

python-cheatsheet / README.md

gto76 Print, Input, Command line argum... c95e741 · 3 days ago History

3578 lines (3050 loc) · 139 KB

Comprehensive Python Cheatsheet

Download text file, Buy PDF, Fork me on GitHub or Check out FAQ.

Contents

1. Collections: `List`, `Dictionary`, `Set`, `Tuple`, `Range`, `Enumerate`, `Iterator`, `Generator`.
2. Types: `Type`, `String`, `Regular_Exp`, `Format`, `Numbers`, `Combinatorics`, `Datetime`.
3. Syntax: `Args`, `Inline`, `Import`, `Decorator`, `Class`, `Duck_Types`, `Enum`, `Exception`.
4. System: `Exit`, `Print`, `Input`, `Command_Line_Arguments`, `Open`, `Path`, `OS_Commands`.
5. Data: `JSON`, `Pickle`, `CSV`, `SQLite`, `Bytes`, `Struct`, `Array`, `Memory_View`, `Deque`.
6. Advanced: `Threading`, `Operator`, `Match_Stmt`, `Logging`, `Introspection`, `Coroutines`.
7. Libraries: `Progress_Bar`, `Plots`, `Tables`, `Curses`, `GUIs`, `Scraping`, `Web`, `Profiling`.
8. Multimedia: `NumPy`, `Image`, `Animation`, `Audio`, `Synthesizer`, `Pygame`, `Pandas`, `Plotly`.

Main

```
if __name__ == '__main__':      # Runs main() if file wasn't i
    main()
```

List

```
<list> = <list>[<slice>]        # Or: <list>[from_inclusive :
```

```
<list>.append(<el>)             # Or: <list> += [<el>]
<list>.extend(<collection>)     # Or: <list> += <collection>
```

```
<list>.sort()                    # Sorts in ascending order.
<list>.reverse()                 # Reverses the list in-place.
<list> = sorted(<collection>)    # Returns a new sorted list.
<iter> = reversed(<list>)        # Returns reversed iterator.
```

```
sum_of_elements  = sum(<collection>)
elementwise_sum  = [sum(pair) for pair in zip(list_a, list_b)]
sorted_by_second = sorted(<collection>, key=lambda el: el[1])
sorted_by_both   = sorted(<collection>, key=lambda el: (el[1],
flatter_list     = list(itertools.chain.from_iterable(<list>))
product_of_elems = functools.reduce(lambda out, el: out * el,
list_of_chars    = list(<str>)
```

- **For details about sorted(), min() and max() see sortable.**
- **Module operator provides functions itemgetter() and mul() that offer the same functionality as lambda expressions above.**

```
<list>.insert(<int>, <el>)       # Inserts item at index and mo
<el>  = <list>.pop([<int>])      # Removes and returns item at
<int> = <list>.count(<el>)       # Returns number of occurrence
<int> = <list>.index(<el>)       # Returns index of the first o
<list>.remove(<el>)              # Removes first occurrence of
<list>.clear()                   # Removes all items. Also work
```

Dictionary

```
<view> = <dict>.keys()                          # Coll. of key
<view> = <dict>.values()                        # Coll. of val
<view> = <dict>.items()                         # Coll. of key
```

```
value  = <dict>.get(key, default=None)          # Returns defa
value  = <dict>.setdefault(key, default=None)   # Returns and
<dict> = collections.defaultdict(<type>)        # Returns a di
<dict> = collections.defaultdict(lambda: 1)     # Returns a di
```

```
<dict> = dict(<collection>)                      # Creates a di
<dict> = dict(zip(keys, values))                 # Creates a di
<dict> = dict.fromkeys(keys [, value])           # Creates a di
```

```
<dict>.update(<dict>)                            # Adds items.
value = <dict>.pop(key)                          # Removes item
{k for k, v in <dict>.items() if v == value}     # Returns set
{k: v for k, v in <dict>.items() if k in keys}   # Returns a di
```

Counter

```
>>> from collections import Counter
>>> colors = ['blue', 'blue', 'blue', 'red', 'red']
>>> counter = Counter(colors)
>>> counter['yellow'] += 1
Counter({'blue': 3, 'red': 2, 'yellow': 1})
>>> counter.most_common()[0]
('blue', 3)
```

Set

```
<set> = set()                                    # `{}` returns
```

```
<set>.add(<el>)                                  # Or: <set> |=
<set>.update(<collection> [, ...])               # Or: <set> |=
```

```
<set>  = <set>.union(<coll.>)                    # Or: <set> |
<set>  = <set>.intersection(<coll.>)             # Or: <set> &
<set>  = <set>.difference(<coll.>)               # Or: <set> -
<set>  = <set>.symmetric_difference(<coll.>)     # Or: <set> ^
<bool> = <set>.issubset(<coll.>)                 # Or: <set> <=
<bool> = <set>.issuperset(<coll.>)               # Or: <set> >=
```

```
<el> = <set>.pop()                                  # Raises KeyEr
<set>.remove(<el>)                                  # Raises KeyEr
<set>.discard(<el>)                                 # Doesn't rais
```

Frozen Set

- **Is immutable and hashable.**
- **That means it can be used as a key in a dictionary or as an element in a set.**

```
<frozenset> = frozenset(<collection>)
```

Tuple

Tuple is an immutable and hashable list.

```
<tuple> = ()                                # Empty tuple.
<tuple> = (<el>,)                           # Or: <el>,
<tuple> = (<el_1>, <el_2> [, ...])          # Or: <el_1>, <el_2
```

Named Tuple

Tuple's subclass with named elements.

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

Range

Immutable and hashable sequence of integers.

```
<range> = range(stop)                       # range(to_exclusiv
<range> = range(start, stop)                # range(from_inclus
<range> = range(start, stop, ±step)         # range(from_inclus
```

```
>>> [i for i in range(3)]
[0, 1, 2]
```

Enumerate

```
for i, el in enumerate(<collection> [, i_start]):
    ...
```

Iterator

```
<iter> = iter(<collection>)                 # `iter(<iter>)` re
<iter> = iter(<function>, to_exclusive)     # A sequence of ret
<el>   = next(<iter> [, default])           # Raises StopIterat
<list> = list(<iter>)                       # Returns a list of
```

Itertools

```
import itertools as it
```

```
<iter> = it.count(start=0, step=1)          # Returns updated v
<iter> = it.repeat(<el> [, times])          # Returns element e
<iter> = it.cycle(<collection>)             # Repeats the seque
```

```
<iter> = it.chain(<coll>, <coll> [, ...])  # Empties collectic
<iter> = it.chain.from_iterable(<coll>)    # Empties collectic
```

```
<iter> = it.islice(<coll>, to_exclusive)   # Only returns firs
<iter> = it.islice(<coll>, from_inc, …)    # `to_exclusive, +s
```

Generator

- **Any function that contains a yield statement returns a generator.**
- **Generators and iterators are interchangeable.**

```
def count(start, step):
    while True:
        yield start
        start += step
```

```
>>> counter = count(10, 2)
>>> next(counter), next(counter), next(counter)
(10, 12, 14)
```

Type

- **Everything is an object.**
- **Every object has a type.**
- **Type and class are synonymous.**

```
<type> = type(<el>)                          # Or: <el>.__clas
<bool> = isinstance(<el>, <type>)            # Or: issubclass(
```

```
>>> type('a'), 'a'.__class__, str
(<class 'str'>, <class 'str'>, <class 'str'>)
```

Some types do not have built-in names, so they must be imported:

```
from types import FunctionType, MethodType, LambdaType, Genera
```

Abstract Base Classes

Each abstract base class specifies a set of virtual subclasses. These classes are then recognized by isinstance() and issubclass() as subclasses of the ABC, although they are really not. ABC can also manually decide whether or not a specific class is its virtual subclass, usually based on which methods the class has implemented. For instance, Iterable ABC looks for method iter(), while Collection ABC looks for iter(), contains() and len().

```
>>> from collections.abc import Iterable, Collection, Sequence
>>> isinstance([1, 2, 3], Iterable)
True
```

```
+------------------+------------+------------+------------+
|                  |  Iterable  | Collection |  Sequence  |
+------------------+------------+------------+------------+
| list, range, str |    yes     |    yes     |    yes     |
| dict, set        |    yes     |    yes     |            |
| iter             |    yes     |            |            |
+------------------+------------+------------+------------+
```

```
>>> from numbers import Number, Complex, Real, Rational, Integ
>>> isinstance(123, Number)
True
```

```
+--------------------+----------+----------+----------+----
------+----------+
|                    |  Number  |  Complex |   Real   |
Rational | Integral |
+--------------------+----------+----------+----------+----
------+----------+
| int                |   yes    |   yes    |   yes    |
yes    |   yes    |
| fractions.Fraction |   yes    |   yes    |   yes    |
yes    |          |
| float              |   yes    |   yes    |   yes    |
|          |
| complex            |   yes    |   yes    |          |
|          |
| decimal.Decimal    |   yes    |          |          |
|          |
+--------------------+----------+----------+----------+----
------+----------+
```

String

Immutable sequence of characters.

```
<str>  = <str>.strip()                        # Strips all whit
<str>  = <str>.strip('<chars>')               # Strips passed c
```

```
<list> = <str>.split()                        # Splits on one c
<list> = <str>.split(sep=None, maxsplit=-1)   # Splits on 'sep'
<list> = <str>.splitlines(keepends=False)     # On [\n\r\f\v\x1
<str>  = <str>.join(<coll_of_strings>)        # Joins elements
```

```
<bool> = <sub_str> in <str>                   # Checks if strin
<bool> = <str>.startswith(<sub_str>)          # Pass tuple of s
<int>  = <str>.find(<sub_str>)                # Returns start i
<int>  = <str>.index(<sub_str>)               # Same, but raise
```

```
<str>  = <str>.lower()                        # Changes the cas
<str>  = <str>.replace(old, new [, count])    # Replaces 'old'
<str>  = <str>.translate(<table>)             # Use `str.maketr
```

```
<str>  = chr(<int>)                           # Converts int to
<int>  = ord(<str>)                           # Converts Unicod
```

- Use `'unicodedata.normalize("NFC", <str>)'` on strings like `'Motörhead'` before comparing them to other strings, because `'ö'` can be stored as one or two characters.
- `'NFC'` converts such characters to a single character, while `'NFD'` converts them to two.

Property Methods

```
<bool> = <str>.isdecimal()                    # Checks for [0-9
<bool> = <str>.isdigit()                      # Checks for [²³¹
<bool> = <str>.isnumeric()                    # Checks for [¼½³
<bool> = <str>.isalnum()                      # Checks for [a-z
<bool> = <str>.isprintable()                  # Checks for [ !#
<bool> = <str>.isspace()                      # Checks for [ \t
```

Regex

Functions for regular expression matching.

```
import re
<str>   = re.sub(<regex>, new, text, count=0)  # Substitutes a
<list>  = re.findall(<regex>, text)            # Returns all c
<list>  = re.split(<regex>, text, maxsplit=0)  # Add brackets
<Match> = re.search(<regex>, text)             # First occurre
<Match> = re.match(<regex>, text)              # Searches only
<iter>  = re.finditer(<regex>, text)           # Returns all c
```

- Argument 'new' can be a function that accepts a Match object and returns a string.
- Argument `'flags=re.IGNORECASE'` can be used with all functions.
- Argument `'flags=re.MULTILINE'` makes `'^'` and `'$'` match the start/end of each line.
- Argument `'flags=re.DOTALL'` makes `'.'` also accept the `'\n'`.
- Use `r'\1'` or `'\\1'` for backreference (`'\1'` returns a character with octal code 1).
- Add `'?'` after `'*'` and `'+'` to make them non-greedy.
- `'re.compile(<regex>)'` returns a Pattern object with methods sub(), findall(), ...

Match Object

```
<str>   = <Match>.group()                    # Returns the w
<str>   = <Match>.group(1)                   # Returns part
<tuple> = <Match>.groups()                   # Returns all b
<int>   = <Match>.start()                    # Returns start
<int>   = <Match>.end()                      # Returns exclu
```

Special Sequences

```
'\d' == '[0-9]'                              # Also [۰-٩…].
'\w' == '[a-zA-Z0-9_]'                       # Also [ª²³…].
'\s' == '[ \t\n\r\f\v]'                      # Also [\x1c-\x
```

- By default, decimal characters, alphanumerics and whitespaces from all alphabets are matched unless `'flags=re.ASCII'` argument is used.
- It restricts special sequence matches to `'[\x00-\x7f]'` (the first 128 characters) and also prevents `'\s'` from accepting `'[\x1c-\x1f]'` (the so-called separator characters).
- Use a capital letter for negation (all non-ASCII characters will be matched when used in combination with ASCII flag).

Format

```
<str> = f'{<el_1>}, {<el_2>}'            # Curly brackets can
<str> = '{}, {}'.format(<el_1>, <el_2>)  # Or: '{0}, {a}'.form
<str> = '%s, %s' % (<el_1>, <el_2>)      # Redundant and infe
```

Example

```
>>> Person = collections.namedtuple('Person', 'name height')
>>> person = Person('Jean-Luc', 187)
>>> f'{person.name} is {person.height / 100} meters tall.'
'Jean-Luc is 1.87 meters tall.'
```

General Options

```
{<el>:<10}                              # '<el>      '
{<el>:^10}                              # '   <el>   '
{<el>:>10}                              # '      <el>'
{<el>:.<10}                             # '<el>......'
{<el>:0}                                # '<el>'
```

- Objects are rendered using `'format(<el>, <options>)'`.
- Options can be generated dynamically: `f'{<el>:{<str/int>}[…]}'`.
- Adding `'='` to the expression prepends it to the output: `f'{1+1=}'` returns `'1+1=2'`.
- Adding `'!r'` to the expression converts object to string by calling its repr() method.

Strings

```
{'abcde':10}                              # 'abcde     '
{'abcde':10.3}                            # 'abc       '
{'abcde':.3}                              # 'abc'
{'abcde'!r:10}                            # "'abcde'   "
```

Numbers

```
{123456:10}                               # '    123456'
{123456:10,}                              # '   123,456'
{123456:10_}                              # '   123_456'
{123456:+10}                              # '   +123456'
{123456:=+10}                             # '+   123456'
{123456: }                                # ' 123456'
{-123456: }                               # '-123456'
```

Floats

```
{1.23456:10.3}                            # '      1.23'
{1.23456:10.3f}                           # '     1.235'
{1.23456:10.3e}                           # ' 1.235e+00'
{1.23456:10.3%}                           # '  123.456%'
```

Comparison of presentation types:

```
+--------------+----------------+----------------+---------
-------+----------------+
|              |    {<float>}   |   {<float>:f}  |
{<float>:e}  |   {<float>:%}  |
+--------------+----------------+----------------+---------
-------+----------------+
|  0.000056789 |   '5.6789e-05' |    '0.000057'  |
'5.678900e-05' |    '0.005679%' |
|  0.00056789  |   '0.00056789' |    '0.000568'  |
'5.678900e-04' |    '0.056789%' |
|  0.0056789   |   '0.0056789'  |    '0.005679'  |
'5.678900e-03' |    '0.567890%' |
|  0.056789    |   '0.056789'   |    '0.056789'  |
'5.678900e-02' |    '5.678900%' |
|  0.56789     |   '0.56789'    |    '0.567890'  |
'5.678900e-01' |   '56.789000%' |
|  5.6789      |   '5.6789'     |    '5.678900'  |
'5.678900e+00' |  '567.890000%' |
| 56.789       |  '56.789'      |   '56.789000'  |
'5.678900e+01' | '5678.900000%' |
+--------------+----------------+----------------+---------
-------+----------------+
```

```
+--------------+----------------+----------------+---------
-------+----------------+
|              |  {<float>:.2}  |  {<float>:.2f} |
{<float>:.2e} |  {<float>:.2%} |
+--------------+----------------+----------------+---------
-------+----------------+
|  0.000056789 |    '5.7e-05'   |      '0.00'    |
'5.68e-05'   |      '0.01%'   |
|  0.00056789  |    '0.00057'   |      '0.00'    |
'5.68e-04'   |      '0.06%'   |
|  0.0056789   |    '0.0057'    |      '0.01'    |
'5.68e-03'   |      '0.57%'   |
|  0.056789    |    '0.057'     |      '0.06'    |
'5.68e-02'   |      '5.68%'   |
|  0.56789     |    '0.57'      |      '0.57'    |
'5.68e-01'   |     '56.79%'   |
|  5.6789      |    '5.7'       |      '5.68'    |
'5.68e+00'   |    '567.89%'   |
| 56.789       |    '5.7e+01'   |     '56.79'    |
'5.68e+01'   |   '5678.90%'   |
+--------------+----------------+----------------+---------
-------+----------------+
```

- `'{<float>:g}'` is `'{<float>:.6}'` with stripped zeros, exponent starting at `'1e+06'`.
- When both rounding up and rounding down are possible, the one that returns result with even last digit is chosen. That makes `'{6.5:.0f}'` a `'6'` and `'{7.5:.0f}'` an `'8'`.
- This rule only effects numbers that can be represented exactly by a float (`.5`, `.25`, ...).

Ints

```
{90:c}                                    # 'Z'
{90:b}                                    # '1011010'
{90:X}                                    # '5A'
```

Numbers

```
<int>      = int(<float/str/bool>)                # Or: math.f
<float>    = float(<int/str/bool>)                # Or: <int/f
<complex>  = complex(real=0, imag=0)              # Or: <int/f
<Fraction> = fractions.Fraction(0, 1)             # Or: Fracti
<Decimal>  = decimal.Decimal(<str/int>)           # Or: Decima
```

- `'int(<str>)'` and `'float(<str>)'` raise ValueError on malformed strings.
- Decimal numbers are stored exactly, unlike most floats where `'1.1 + 2.2 != 3.3'`.
- Floats can be compared with: `'math.isclose(<float>, <float>)'`.
- Precision of decimal operations is set with: `'decimal.getcontext().prec = <int>'`.

Basic Functions

```
<num> = pow(<num>, <num>)                         # Or: <num>
<num> = abs(<num>)                                # <float> =
<num> = round(<num> [, ±ndigits])                 # `round(12
```

Math

```
from math import e, pi, inf, nan, isinf, isnan    # `<el> == r
from math import sin, cos, tan, asin, acos, atan  # Also: degr
from math import log, log10, log2                 # Log can ac
```

Statistics

```
from statistics import mean, median, variance     # Also: stde
```

Random

```
from random import random, randint, choice          # Also: shuf
<float> = random()                                  # A float in
<int>   = randint(from_inc, to_inc)                 # An int ins
<el>    = choice(<sequence>)                        # Keeps the
```

Bin, Hex

```
<int> = ±0b<bin>                                    # Or: ±0x<he
<int> = int('±<bin>', 2)                            # Or: int('±
<int> = int('±0b<bin>', 0)                          # Or: int('±
<str> = bin(<int>)                                  # Returns '[
```

Bitwise Operators

```
<int> = <int> & <int>                               # And (0b116
<int> = <int> | <int>                               # Or  (0b116
<int> = <int> ^ <int>                               # Xor (0b116
<int> = <int> << n_bits                             # Left shift
<int> = ~<int>                                      # Not. Also
```

Combinatorics

```
import itertools as it
```

```
>>> list(it.product([0, 1], repeat=3))
[(0, 0, 0), (0, 0, 1), (0, 1, 0), (0, 1, 1),
 (1, 0, 0), (1, 0, 1), (1, 1, 0), (1, 1, 1)]
```

```
>>> list(it.product('abc', 'abc'))                    #   a  b
[('a', 'a'), ('a', 'b'), ('a', 'c'),                  # a x  x
 ('b', 'a'), ('b', 'b'), ('b', 'c'),                  # b x  x
 ('c', 'a'), ('c', 'b'), ('c', 'c')]                  # c x  x
```

```
>>> list(it.combinations('abc', 2))                   #   a  b
[('a', 'b'), ('a', 'c'),                              # a .  x
 ('b', 'c')]                                          # b .  .
```

```
>>> list(it.combinations_with_replacement('abc', 2))  #   a  b
[('a', 'a'), ('a', 'b'), ('a', 'c'),                  # a x  x
 ('b', 'b'), ('b', 'c'),                              # b .  x
 ('c', 'c')]                                          # c .  .
```

```
>>> list(it.permutations('abc', 2))                   #   a  b
[('a', 'b'), ('a', 'c'),                              # a .  x
 ('b', 'a'), ('b', 'c'),                              # b x  .
 ('c', 'a'), ('c', 'b')]                              # c x  x
```

Datetime

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

```
# $ pip3 install python-dateutil
from datetime import date, time, datetime, timedelta, timezone
from dateutil.tz import tzlocal, gettz
```

```
<D>  = date(year, month, day)                # Only accepts val
<T>  = time(hour=0, minute=0, second=0)      # Also: `microsecc
<DT> = datetime(year, month, day, hour=0)    # Also: `minute=0,
<TD> = timedelta(weeks=0, days=0, hours=0)   # Also: `minutes=(
```

- Aware `<a>` time and datetime objects have defined timezone, while naive `<n>` don't. If object is naive, it is presumed to be in the system's timezone!
- `'fold=1'` means the second pass in case of time jumping back for one hour.
- Timedelta normalizes arguments to ±days, seconds (< 86 400) and microseconds (< 1M).
- Use `'<D/DT>.weekday()'` to get the day of the week as an int, with Monday being 0.

Now

```
<D/DTn>  = D/DT.today()                    # Current local da
<DTa>    = DT.now(<tzinfo>)                # Aware DT from cu
```

- To extract time use `'<DTn>.time()'`, `'<DTa>.time()'` or `'<DTa>.timetz()'`.

Timezone

```
<tzinfo> = timezone.utc                    # London without d
<tzinfo> = timezone(<timedelta>)           # Timezone with fi
<tzinfo> = tzlocal()                       # Local tz with dy
<tzinfo> = gettz('<Continent>/<City>')     # 'Continent/City_
<DTa>    = <DT>.astimezone([<tzinfo>])     # Converts DT to t
<Ta/DTa> = <T/DT>.replace(tzinfo=<tzinfo>) # Changes object's
```

- Timezones returned by gettz(), tzlocal(), and implicit local timezone of naive objects have offsets that vary through time due to DST and historical changes of the zone's base offset.
- Standard library's zoneinfo.ZoneInfo() can be used instead of gettz() on Python 3.9 and later. It requires 'tzdata' package on Windows. It doesn't return local tz if arg. is omitted.

Encode

```
<D/T/DT> = D/T/DT.fromisoformat(<str>)      # Object from ISO
<DT>     = DT.strptime(<str>, '<format>')   # Datetime from st
<D/DTn>  = D/DT.fromordinal(<int>)          # D/DTn from days
<DTn>    = DT.fromtimestamp(<float>)        # Local time DTn f
<DTa>    = DT.fromtimestamp(<float>, <tz>)  # Aware datetime f
```

- ISO strings come in following forms: `'YYYY-MM-DD'`, `'HH:MM:SS.mmmuuu[±HH:MM]'`, or both separated by an arbitrary character. All parts following the hours are optional.
- Python uses the Unix Epoch: `'1970-01-01 00:00 UTC'`, `'1970-01-01 01:00 CET'`, ...

Decode

```
<str>    = <D/T/DT>.isoformat(sep='T')      # Also `timespec='
<str>    = <D/T/DT>.strftime('<format>')    # Custom string re
<int>    = <D/DT>.toordinal()               # Days since Gregc
<float>  = <DTn>.timestamp()                # Seconds since th
<float>  = <DTa>.timestamp()                # Seconds since th
```

Format

```
>>> dt = datetime.strptime('2025-08-14 23:39:00.00 +0200', '%Y
>>> dt.strftime("%dth of %B '%y (%a), %I:%M %p %Z")
"14th of August '25 (Thu), 11:39 PM UTC+02:00"
```

- `'%z'` accepts `'±HH[:]MM'` and returns `'±HHMM'` or empty string if datetime is naive.
- `'%Z'` accepts `'UTC/GMT'` and local timezone's code and returns timezone's name, `'UTC[±HH:MM]'` if timezone is nameless, or an empty string if datetime is naive.

Arithmetics

```
<bool>   = <D/T/DTn> > <D/T/DTn>          # Ignores time jun
<bool>   = <DTa>     > <DTa>              # Ignores time jun
<TD>     = <D/DTn>   - <D/DTn>            # Ignores jumps. (
<TD>     = <DTa>     - <DTa>              # Ignores time jun
<D/DT>   = <D/DT>    ± <TD>               # Returned datetin
<TD>     = <TD>      * <float>            # Also: <TD> = abs
<float>  = <TD>      / <TD>               # How many hours/v
```

Arguments

Inside Function Call

```
func(<positional_args>)                          # func(0, 0)
func(<keyword_args>)                             # func(x=0,
func(<positional_args>, <keyword_args>)          # func(0, y=
```

Inside Function Definition

```
def func(<nondefault_args>): ...                 # def func(
def func(<default_args>): ...                    # def func(
def func(<nondefault_args>, <default_args>): ...  # def func(
```

- **Default values are evaluated when function is first encountered in the scope.**
- **Any mutation of a mutable default value will persist between invocations!**

Splat Operator

Inside Function Call

Splat expands a collection into positional arguments, while splatty-splat expands a dictionary into keyword arguments.

```
args   = (1, 2)
kwargs = {'x': 3, 'y': 4, 'z': 5}
func(*args, **kwargs)
```

Is the same as:

```
func(1, 2, x=3, y=4, z=5)
```

Inside Function Definition

Splat combines zero or more positional arguments into a tuple, while splatty-splat combines zero or more keyword arguments into a dictionary.

```
def add(*a):
    return sum(a)
```

```
>>> add(1, 2, 3)
6
```

Legal argument combinations:

```
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,
```

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

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

Other Uses

```
<list>  = [*<coll.> [, ...]]    # Or: list(<collection>) [+ ..
<tuple> = (*<coll.>, [...])     # Or: tuple(<collection>) [+ .
<set>   = {*<coll.> [, ...]}    # Or: set(<collection>) [| ...
<dict>  = {**<dict> [, ...]}    # Or: dict(<dict>) [| ...] (si
```

```
head, *body, tail = <coll.>     # Head or tail can be omitted.
```

Inline

Lambda

```
<func> = lambda: <return_value>                     # A single
<func> = lambda <arg_1>, <arg_2>: <return_value>    # Also acc
```

Comprehensions

```
<list> = [i+1 for i in range(10)]                   # Or: [1,
<iter> = (i for i in range(10) if i > 5)            # Or: iter
<set>  = {i+5 for i in range(10)}                   # Or: {5,
<dict> = {i: i*2 for i in range(10)}                # Or: {0:
```

```
>>> [l+r for l in 'abc' for r in 'abc']
['aa', 'ab', 'ac', ..., 'cc']
```

Map, Filter, Reduce

```
from functools import reduce
```

```
<iter> = map(lambda x: x + 1, range(10))            # Or: itei
<iter> = filter(lambda x: x > 5, range(10))         # Or: itei
<obj>  = reduce(lambda out, x: out + x, range(10))  # Or: 45
```

Any, All

```
<bool> = any(<collection>)                          # Is `bool
<bool> = all(<collection>)                          # Is True
```

Conditional Expression

```
<obj> = <exp> if <condition> else <exp>             # Only one
```

```
>>> [a if a else 'zero' for a in (0, 1, 2, 3)]      # `any([0,
['zero', 1, 2, 3]
```

Named Tuple, Enum, Dataclass

```
from collections import namedtuple
Point = namedtuple('Point', 'x y')                  # Creates
point = Point(0, 0)                                 # Returns
```

```
from enum import Enum
Direction = Enum('Direction', 'N E S W')            # Creates
direction = Direction.N                             # Returns
```

```
from dataclasses import make_dataclass
Player = make_dataclass('Player', ['loc', 'dir'])   # Creates
player = Player(point, direction)                   # Returns
```

Imports

```
import <module>            # Imports a built-in or '<module>.p
import <package>           # Imports a built-in or '<package>/
import <package>.<module>  # Imports a built-in or '<package>/
```

- Package is a collection of modules, but it can also define its own objects.
- On a filesystem this corresponds to a directory of Python files with an optional init script.
- Running `'import <package>'` does not automatically provide access to the package's modules unless they are explicitly imported in its init script.

Closure

We have/get a closure in Python when:

- A nested function references a value of its enclosing function and then
- the enclosing function returns the nested function.

```
def get_multiplier(a):
    def out(b):
        return a * b
    return out
```

```
>>> multiply_by_3 = get_multiplier(3)
>>> multiply_by_3(10)
30
```

- If multiple nested functions within enclosing function reference the same value, that value gets shared.
- To dynamically access function's first free variable use `'<function>.__closure__[0].cell_contents'`.

Partial

```
from functools import partial
<function> = partial(<function> [, <arg_1>, <arg_2>, ...])
```

```
>>> def multiply(a, b):
...     return a * b
>>> multiply_by_3 = partial(multiply, 3)
>>> multiply_by_3(10)
30
```

- Partial is also useful in cases when function needs to be passed as an argument because it enables us to set its arguments beforehand.
- A few examples being: `'defaultdict(<func>)'`, `'iter(<func>, to_exc)'` and dataclass's `'field(default_factory=<func>)'`.

Non-Local

If variable is being assigned to anywhere in the scope, it is regarded as a local variable, unless it is declared as a 'global' or a 'nonlocal'.

```python
def get_counter():
    i = 0
    def out():
        nonlocal i
        i += 1
        return i
    return out
```

```python
>>> counter = get_counter()
>>> counter(), counter(), counter()
(1, 2, 3)
```

Decorator

- A decorator takes a function, adds some functionality and returns it.
- It can be any callable, but is usually implemented as a function that returns a closure.

```python
@decorator_name
def function_that_gets_passed_to_decorator():
    ...
```

Debugger Example

Decorator that prints function's name every time the function is called.

```python
from functools import wraps

def debug(func):
    @wraps(func)
    def out(*args, **kwargs):
        print(func.__name__)
        return func(*args, **kwargs)
    return out

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

- Wraps is a helper decorator that copies the metadata of the passed function (func) to the function it is wrapping (out).
- Without it, `'add.__name__'` would return `'out'`.

LRU Cache

Decorator that caches function's return values. All function's arguments must be hashable.

```python
from functools import lru_cache

@lru_cache(maxsize=None)
def fib(n):
    return n if n < 2 else fib(n-2) + fib(n-1)
```

- Default size of the cache is 128 values. Passing `'maxsize=None'` makes it unbounded.
- CPython interpreter limits recursion depth to 1000 by default. To increase it use `'sys.setrecursionlimit(<depth>)'`.

Parametrized Decorator

A decorator that accepts arguments and returns a normal decorator that accepts a function.

```python
from functools import wraps

def debug(print_result=False):
    def decorator(func):
        @wraps(func)
        def out(*args, **kwargs):
            result = func(*args, **kwargs)
            print(func.__name__, result if print_result else '
            return result
        return out
    return decorator

@debug(print_result=True)
def add(x, y):
    return x + y
```

- Using only `'@debug'` to decorate the add() function would not work here, because debug would then receive the add() function as a 'print_result' argument. Decorators can however manually check if the argument they received is a function and act accordingly.

Class

```python
class <name>:
    def __init__(self, a):
        self.a = a
    def __repr__(self):
        class_name = self.__class__.__name__
        return f'{class_name}({self.a!r})'
    def __str__(self):
        return str(self.a)

    @classmethod
    def get_class_name(cls):
        return cls.__name__
```

- Return value of repr() should be unambiguous and of str() readable.
- If only repr() is defined, it will also be used for str().

- Methods decorated with `'@staticmethod'` do not receive 'self' nor 'cls' as their first arg.

Expressions that call the str() method:

```
print(<el>)
f'{<el>}'
logging.warning(<el>)
csv.writer(<file>).writerow([<el>])
raise Exception(<el>)
```

Expressions that call the repr() method:

```
print/str/repr([<el>])
print/str/repr({<el>: <el>})
f'{<el>!r}'
Z = dataclasses.make_dataclass('Z', ['a']); print/str/repr(Z(<
>>> <el>
```

Constructor Overloading

```
class <name>:
    def __init__(self, a=None):
        self.a = a
```

Inheritance

```
class Person:
    def __init__(self, name):
        self.name = name

class Employee(Person):
    def __init__(self, name, staff_num):
        super().__init__(name)
        self.staff_num = staff_num
```

Multiple Inheritance

```
class A: pass
class B: pass
class C(A, B): pass
```

MRO determines the order in which parent classes are traversed when searching for a method or an attribute:

```
>>> C.mro()
[<class 'C'>, <class 'A'>, <class 'B'>, <class 'object'>]
```

Property

Pythonic way of implementing getters and setters.

```
class Person:
    @property
    def name(self):
        return ' '.join(self._name)

    @name.setter
    def name(self, value):
        self._name = value.split()
```

```
>>> person = Person()
>>> person.name = '\t Guido  van Rossum \n'
>>> person.name
'Guido van Rossum'
```

Dataclass

Decorator that automatically generates init(), repr() and eq() special methods.

```
from dataclasses import dataclass, field

@dataclass(order=False, frozen=False)
class <class_name>:
    <attr_name>: <type>
    <attr_name>: <type> = <default_value>
    <attr_name>: list/dict/set = field(default_factory=list/di
```

- Objects can be made sortable with `'order=True'` and immutable with `'frozen=True'`.
- For object to be hashable, all attributes must be hashable and 'frozen' must be True.
- Function field() is needed because `'<attr_name>: list = []'` would make a list that is shared among all instances. Its 'default_factory' argument can be any callable.
- For attributes of arbitrary type use `'typing.Any'`.

Inline:

```
from dataclasses import make_dataclass
<class> = make_dataclass('<class_name>', <coll_of_attribute_na
<class> = make_dataclass('<class_name>', <coll_of_tuples>)
<tuple> = ('<attr_name>', <type> [, <default_value>])
```

Rest of type annotations (CPython interpreter ignores them all):

```
import collections.abc as abc, typing as tp
<var_name>: list/set/abc.Iterable/abc.Sequence/tp.Optional[<ty
<var_name>: dict/tuple/tp.Union[<type>, ...] [= <obj>]
def func(<arg_name>: <type> [= <obj>]) -> <type>: ...
```

Slots

Mechanism that restricts objects to attributes listed in 'slots' and significantly reduces their memory footprint.

```
class MyClassWithSlots:
    __slots__ = ['a']
    def __init__(self):
        self.a = 1
```

Copy

```
from copy import copy, deepcopy
<object> = copy(<object>)
<object> = deepcopy(<object>)
```

Duck Types

A duck type is an implicit type that prescribes a set of special methods. Any object that has those methods defined is considered a member of that duck type.

Comparable

- If eq() method is not overridden, it returns `'id(self) == id(other)'`, which is the same as `'self is other'`.
- That means all objects compare not equal by default.
- Only the left side object has eq() method called, unless it returns NotImplemented, in which case the right object is consulted. False is returned if both return NotImplemented.
- Ne() automatically works on any object that has eq() defined.

```
class MyComparable:
    def __init__(self, a):
        self.a = a
    def __eq__(self, other):
        if isinstance(other, type(self)):
            return self.a == other.a
        return NotImplemented
```

Hashable

- Hashable object needs both hash() and eq() methods and its hash value should never change.
- Hashable objects that compare equal must have the same hash value, meaning default hash() that returns `'id(self)'` will not do.
- That is why Python automatically makes classes unhashable if you only implement eq().

```
class MyHashable:
    def __init__(self, a):
        self._a = a
    @property
    def a(self):
        return self._a
    def __eq__(self, other):
        if isinstance(other, type(self)):
            return self.a == other.a
        return NotImplemented
    def __hash__(self):
        return hash(self.a)
```

Sortable

- With 'total_ordering' decorator, you only need to provide eq() and one of lt(), gt(), le() or ge() special methods and the rest will be automatically generated.
- Functions sorted() and min() only require lt() method, while max() only requires gt(). However, it is best to define them all so that confusion doesn't arise in other contexts.
- When two lists, strings or dataclasses are compared, their values get compared in order until a pair of unequal values is found. The comparison of this two values is then returned. The shorter sequence is considered smaller in case of all values being equal.
- For proper alphabetical order pass `'key=locale.strxfrm'` to sorted() after running `'locale.setlocale(locale.LC_COLLATE, "en_US.UTF-`

`8")'`.

```
from functools import total_ordering

@total_ordering
class MySortable:
    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
```

Iterator

- Any object that has methods next() and iter() is an iterator.
- Next() should return next item or raise StopIteration exception.
- Iter() should return 'self'.

```
class Counter:
    def __init__(self):
        self.i = 0
    def __next__(self):
        self.i += 1
        return self.i
    def __iter__(self):
        return self
```

```
>>> counter = Counter()
>>> next(counter), next(counter), next(counter)
(1, 2, 3)
```

Python has many different iterator objects:

- Sequence iterators returned by the iter() function, such as list_iterator and set_iterator.
- Objects returned by the itertools module, such as count, repeat and cycle.
- Generators returned by the generator functions and generator expressions.
- File objects returned by the open() function, etc.

Callable

- All functions and classes have a call() method, hence are callable.
- When this cheatsheet uses `'<function>'` as an argument, it actually means `'<callable>'`.

```
class Counter:
    def __init__(self):
        self.i = 0
    def __call__(self):
        self.i += 1
        return self.i
```

```
>>> counter = Counter()
>>> counter(), counter(), counter()
(1, 2, 3)
```

Context Manager

- With statements only work with objects that have enter() and exit() special methods.
- Enter() should lock the resources and optionally return an object.
- Exit() should release the resources.
- Any exception that happens inside the with block is passed to the exit() method.
- The exit() method can suppress the exception by returning a true value.

```
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!
```

Iterable Duck Types

Iterable

- Only required method is iter(). It should return an iterator of object's items.
- Contains() automatically works on any object that has iter() defined.

```
class MyIterable:
    def __init__(self, a):
        self.a = a
    def __iter__(self):
        return iter(self.a)
    def __contains__(self, el):
        return el in self.a
```

```
>>> obj = MyIterable([1, 2, 3])
>>> [el for el in obj]
[1, 2, 3]
>>> 1 in obj
True
```

Collection

- Only required methods are iter() and len(). Len() should return the number of items.
- This cheatsheet actually means `'<iterable>'` when it uses `'<collection>'`.
- I chose not to use the name 'iterable' because it sounds scarier and more vague than 'collection'. The only drawback of this decision is that the reader could think a certain function doesn't accept iterators when it does, since iterators are the only built-in objects that are iterable but are not collections.

```
class MyCollection:
    def __init__(self, a):
        self.a = a
    def __iter__(self):
        return iter(self.a)
    def __contains__(self, el):
        return el in self.a
    def __len__(self):
        return len(self.a)
```

Sequence

- Only required methods are getitem() and len().
- Getitem() should return an item at the passed index or raise IndexError.
- Iter() and contains() automatically work on any object that has getitem() defined.
- Reversed() automatically works on any object that has getitem() and

len() defined.

```
class MySequence:
    def __init__(self, a):
        self.a = a
    def __iter__(self):
        return iter(self.a)
    def __contains__(self, el):
        return el in self.a
    def __len__(self):
        return len(self.a)
    def __getitem__(self, i):
        return self.a[i]
    def __reversed__(self):
        return reversed(self.a)
```

Discrepancies between glossary definitions and abstract base classes:

- Glossary defines iterable as any object with iter() or getitem() and sequence as any object with getitem() and len(). It does not define collection.
- Passing ABC Iterable to isinstance() or issubclass() checks whether object/class has method iter(), while ABC Collection checks for iter(), contains() and len().

ABC Sequence

- It's a richer interface than the basic sequence.
- Extending it generates iter(), contains(), reversed(), index() and count().
- Unlike `'abc.Iterable'` and `'abc.Collection'`, it is not a duck type. That is why `'issubclass(MySequence, abc.Sequence)'` would return False even if MySequence had all the methods defined. It however recognizes list, tuple, range, str, bytes, bytearray, array, memoryview and deque, because they are registered as its virtual subclasses.

```python
from collections import abc

class MyAbcSequence(abc.Sequence):
    def __init__(self, a):
        self.a = a
    def __len__(self):
        return len(self.a)
    def __getitem__(self, i):
        return self.a[i]
```

Table of required and automatically available special methods:

```
+------------+------------+------------+------------+------
--------+
|            |  Iterable  | Collection |  Sequence  |
abc.Sequence |
+------------+------------+------------+------------+------
--------+
| iter()     |    REQ     |    REQ     |    Yes     |
Yes      |
| contains() |    Yes     |    Yes     |    Yes     |
Yes      |
| len()      |            |    REQ     |    REQ     |
REQ      |
| getitem()  |            |            |    REQ     |
REQ      |
| reversed() |            |            |    Yes     |
Yes      |
| index()    |            |            |            |
Yes      |
| count()    |            |            |            |
Yes      |
+------------+------------+------------+------------+------
--------+
```

- Other ABCs that generate missing methods are: MutableSequence, Set, MutableSet, Mapping and MutableMapping.
- Names of their required methods are stored in `'<abc>.__abstractmethods__'`.

Enum

```
from enum import Enum, auto
```

```
class <enum_name>(Enum):
    <member_name> = auto()
    <member_name> = <value>
    <member_name> = <value>, <value>
```

- **Function auto() returns an increment of the last numeric value or 1.**
- **Accessing a member named after a reserved keyword causes SyntaxError.**
- **Methods receive the member they were called on as the 'self' argument.**

```
<member> = <enum>.<member_name>           # Returns a member.
<member> = <enum>['<member_name>']        # Returns a member.
<member> = <enum>(<value>)                # Returns a member.
<str>    = <member>.name                  # Returns member's r
<obj>    = <member>.value                 # Returns member's v
```

```
<list>   = list(<enum>)                   # Returns enum's men
<list>   = [a.name for a in <enum>]       # Returns enum's men
<list>   = [a.value for a in <enum>]      # Returns enum's men
<member> = random.choice(list(<enum>))    # Returns a random m
```

```
def get_next_member(member):
    members = list(type(member))
    index = members.index(member) + 1
    return members[index % len(members)]
```

Inline

```
Cutlery = Enum('Cutlery', 'FORK KNIFE SPOON')
Cutlery = Enum('Cutlery', ['FORK', 'KNIFE', 'SPOON'])
Cutlery = Enum('Cutlery', {'FORK': 1, 'KNIFE': 2, 'SPOON': 3})
```

User-defined functions cannot be values, so they must be wrapped:

```
from functools import partial
LogicOp = Enum('LogicOp', {'AND': partial(lambda l, r: l and r),
                           'OR':  partial(lambda l, r: l or r)})
```

Exceptions

```
try:
    <code>
except <exception>:
    <code>
```

Complex Example

```
try:
    <code_1>
except <exception_a>:
    <code_2_a>
except <exception_b>:
    <code_2_b>
else:
    <code_2_c>
finally:
    <code_3>
```

- Code inside the `'else'` block will only be executed if `'try'` block had no exceptions.
- Code inside the `'finally'` block will always be executed (unless a signal is received).

- All variables that are initialized in executed blocks are also visible in all subsequent blocks, as well as outside the try/except clause (only function block delimits scope).
- To catch signals use `'signal.signal(signal_number, <func>)'`.

Catching Exceptions

```
except <exception>: ...
except <exception> as <name>: ...
except (<exception>, [...]): ...
except (<exception>, [...]) as <name>: ...
```

- Also catches subclasses of the exception.
- Use `'traceback.print_exc()'` to print the error message to stderr.
- Use `'print(<name>)'` to print just the cause of the exception (its arguments).
- Use `'logging.exception(<message>)'` to log the passed message, followed by the full error message of the caught exception.

Raising Exceptions

```
raise <exception>
raise <exception>()
raise <exception>(<el> [, ...])
```

Re-raising caught exception:

```
except <exception> [as <name>]:
    ...
    raise
```

Exception Object

```
arguments = <name>.args
exc_type  = <name>.__class__
filename  = <name>.__traceback__.tb_frame.f_code.co_filename
func_name = <name>.__traceback__.tb_frame.f_code.co_name
line      = linecache.getline(filename, <name>.__traceback__.t
trace_str = ''.join(traceback.format_tb(<name>.__traceback__))
error_msg = ''.join(traceback.format_exception(type(<name>), <
```

Built-in Exceptions

```
BaseException
 +-- SystemExit                   # Raised by the
sys.exit() function.
 +-- KeyboardInterrupt            # Raised when the user
hits the interrupt key (ctrl-c).
 +-- Exception                    # User-defined exceptions
should be derived from this class.
      +-- ArithmeticError         # Base class for
arithmetic errors such as ZeroDivisionError.
      +-- AssertionError          # Raised by `assert
<exp>` if expression returns false value.
      +-- AttributeError          # Raised when object
doesn't have requested attribute/method.
      +-- EOFError                # Raised by input() when
it hits an end-of-file condition.
      +-- LookupError             # Base class for errors
when a collection can't find an item.
      |    +-- IndexError         # Raised when a sequence
index is out of range.
      |    +-- KeyError           # Raised when a
dictionary key or set element is missing.
      +-- MemoryError             # Out of memory. Could be
too late to start deleting vars.
      +-- NameError               # Raised when nonexistent
name (variable/func/class) is used.
      |    +-- UnboundLocalError  # Raised when local name
is used before it's being defined.
      +-- OSError                 # Errors such as
FileExistsError/TimeoutError (see #Open).
      |    +-- ConnectionError    # Errors such as
BrokenPipeError/ConnectionAbortedError.
```

```
      +-- RuntimeError            # Raised by errors that
don't fall into other categories.
      |    +-- NotImplementedEr…  # Can be raised by
abstract methods or by unfinished code.
      |    +-- RecursionError     # Raised when the maximum
recursion depth is exceeded.
      +-- StopIteration           # Raised when an empty
iterator is passed to next().
      +-- TypeError               # When an argument of the
wrong type is passed to function.
      +-- ValueError              # When argument has the
right type but inappropriate value.
```

Collections and their exceptions:

```
+-----------+------------+------------+------------+
|           |    List    |    Set     |    Dict    |
+-----------+------------+------------+------------+
| getitem() | IndexError |            |  KeyError  |
| pop()     | IndexError |  KeyError  |  KeyError  |
| remove()  | ValueError |  KeyError  |            |
| index()   | ValueError |            |            |
+-----------+------------+------------+------------+
```

Useful built-in exceptions:

```
raise TypeError('Argument is of the wrong type!')
raise ValueError('Argument has the right type but an inappropı
raise RuntimeError('I am too lazy to define my own exception!'
```

User-defined Exceptions

```
class MyError(Exception): pass
class MyInputError(MyError): pass
```

Exit

Exits the interpreter by raising SystemExit exception.

```
import sys
sys.exit()                          # Exits with exit code 0 (su
sys.exit(<el>)                      # Prints to stderr and exits
sys.exit(<int>)                     # Exits with the passed exit
```

Print

```
print(<el_1>, ..., sep=' ', end='\n', file=sys.stdout, flush=F
```

- Use `'file=sys.stderr'` for messages about errors.
- Stdout and stderr streams hold output in a buffer until they receive a string containing '\n' or '\r', buffer reaches 4096 characters, `'flush=True'` is used, or program exits.

Pretty Print

```
from pprint import pprint
pprint(<collection>, width=80, depth=None, compact=False, sort
```

- Each item is printed on its own line if collection takes up more than 'width' characters.
- Nested collections that are 'depth' levels deep get printed as '...'.

Input

```
<str> = input(prompt=None)
```

- Reads a line from the user input or pipe if present (trailing newline gets stripped).
- Prompt string is printed to the standard output before reading input.

- Raises EOFError when user hits EOF (ctrl-d/ctrl-z⏎) or input stream gets exhausted.

Command Line Arguments

```
import sys
scripts_path = sys.argv[0]
arguments    = sys.argv[1:]
```

Argument Parser

```
from argparse import ArgumentParser, FileType
p = ArgumentParser(description=<str>)
p.add_argument('-<short_name>', '--<name>', action='store_true
p.add_argument('-<short_name>', '--<name>', type=<type>)
p.add_argument('<name>', type=<type>, nargs=1)
p.add_argument('<name>', type=<type>, nargs='+')
p.add_argument('<name>', type=<type>, nargs='*')
<args> = p.parse_args()
<obj>  = <args>.<name>
```

- Use `'help=<str>'` to set argument description that will be displayed in help message.
- Use `'default=<el>'` to set option's default value.
- Use `'type=FileType(<mode>)'` for files. Accepts 'encoding', but 'newline' is None.

Open

Opens the file and returns a corresponding file object.

```
<file> = open(<path>, mode='r', encoding=None, newline=None)
```

- `'encoding=None'` means that the default encoding is used, which is

platform dependent. Best practice is to use `'encoding="utf-8"'` whenever possible.

- `'newline=None'` means all different end of line combinations are converted to '\n' on read, while on write all '\n' characters are converted to system's default line separator.
- `'newline=""'` means no conversions take place, but input is still broken into chunks by readline() and readlines() on every '\n', '\r' and '\r\n'.

Modes

- `'r'` - Read (default).
- `'w'` - Write (truncate).
- `'x'` - Write or fail if the file already exists.
- `'a'` - Append.
- `'w+'` - Read and write (truncate).
- `'r+'` - Read and write from the start.
- `'a+'` - Read and write from the end.
- `'b'` - Binary mode (`'br'`, `'bw'`, `'bx'`, ...).

Exceptions

- `'FileNotFoundError'` can be raised when reading with `'r'` or `'r+'`.
- `'FileExistsError'` can be raised when writing with `'x'`.
- `'IsADirectoryError'` and `'PermissionError'` can be raised by any.
- `'OSError'` is the parent class of all listed exceptions.

File Object

```
<file>.seek(0)                      # Moves to the start of th
<file>.seek(offset)                 # Moves 'offset' chars/byt
<file>.seek(0, 2)                   # Moves to the end of the
<bin_file>.seek(±offset, <anchor>)  # Anchor: 0 start, 1 curre
```

```
<str/bytes> = <file>.read(size=-1)  # Reads 'size' chars/bytes
<str/bytes> = <file>.readline()     # Returns a line or empty
<list>      = <file>.readlines()    # Returns a list of remain
<str/bytes> = next(<file>)          # Returns a line using bu
```

```
<file>.write(<str/bytes>)           # Writes a string or bytes
<file>.writelines(<collection>)     # Writes a coll. of string
<file>.flush()                      # Flushes write buffer. Ru
<file>.close()                      # Closes the file after fl
```

- **Methods do not add or strip trailing newlines, not even writelines().**

Read Text from File

```
def read_file(filename):
    with open(filename, encoding='utf-8') as file:
        return file.readlines()
```

Write Text to File

```
def write_to_file(filename, text):
    with open(filename, 'w', encoding='utf-8') as file:
        file.write(text)
```

Paths

```
import os, glob
from pathlib import Path
```

```
<str>  = os.getcwd()                # Returns the current work
<str>  = os.path.join(<path>, ...)  # Joins two or more pathna
<str>  = os.path.realpath(<path>)   # Resolves symlinks and ca
```

```
<str>  = os.path.basename(<path>)   # Returns final component
<str>  = os.path.dirname(<path>)    # Returns path without the
<tup.> = os.path.splitext(<path>)   # Splits on last period of
```

```
<list> = os.listdir(path='.')       # Returns filenames locate
<list> = glob.glob('<pattern>')     # Returns paths matching t
```

```
<bool> = os.path.exists(<path>)     # Or: <Path>.exists()
<bool> = os.path.isfile(<path>)     # Or: <DirEntry/Path>.is_f
<bool> = os.path.isdir(<path>)      # Or: <DirEntry/Path>.is_d
```

```
<stat> = os.stat(<path>)            # Or: <DirEntry/Path>.stat
<real> = <stat>.st_mtime/st_size/…  # Modification time, size
```

DirEntry

Unlike listdir(), scandir() returns DirEntry objects that cache isfile, isdir and on Windows also stat information, thus significantly increasing the performance of code that requires it.

```
<iter> = os.scandir(path='.')       # Returns DirEntry objects
<str>  = <DirEntry>.path            # Returns the whole path a
<str>  = <DirEntry>.name            # Returns final component
<file> = open(<DirEntry>)           # Opens the file and retur
```

Path Object

```
<Path> = Path(<path> [, ...])       # Accepts strings, Paths a
<Path> = <path> / <path> [/ ...]    # First or second path mus
<Path> = <Path>.resolve()           # Returns absolute path wi
```

```
<Path> = Path()                      # Returns relative cwd. A
<Path> = Path.cwd()                  # Returns absolute cwd. A
<Path> = Path.home()                 # Returns user's home dire
<Path> = Path(__file__).resolve()    # Returns script's path i
```

```
<Path> = <Path>.parent               # Returns Path without the
<str>  = <Path>.name                 # Returns final component
<str>  = <Path>.stem                 # Returns final component
<str>  = <Path>.suffix               # Returns final component'
<tup.> = <Path>.parts                # Returns all components a
```

```
<iter> = <Path>.iterdir()            # Returns directory conter
<iter> = <Path>.glob('<pattern>')    # Returns Paths matching t
```

```
<str>  = str(<Path>)                 # Returns path as a string
<file> = open(<Path>)                # Also <Path>.read/write_t
```

OS Commands

```
import os, shutil, subprocess
```

```
os.chdir(<path>)                     # Changes the current work
os.mkdir(<path>, mode=0o777)         # Creates a directory. Per
os.makedirs(<path>, mode=0o777)      # Creates all path's dirs.
```

```
shutil.copy(from, to)                # Copies the file. 'to' ca
shutil.copy2(from, to)               # Also copies creation and
shutil.copytree(from, to)            # Copies the directory. 't
```

```
os.rename(from, to)                 # Renames/moves the file c
os.replace(from, to)                # Same, but overwrites fil
shutil.move(from, to)               # Rename() that moves intc
```

```
os.remove(<path>)                   # Deletes the file.
os.rmdir(<path>)                    # Deletes the empty direct
shutil.rmtree(<path>)               # Deletes the directory.
```

- **Paths can be either strings, Paths or DirEntry objects.**
- **Functions report OS related errors by raising either OSError or one of its subclasses.**

Shell Commands

```
<pipe> = os.popen('<command>')      # Executes command in sh/c
<str>  = <pipe>.read(size=-1)       # Reads 'size' chars or ur
<int>  = <pipe>.close()             # Closes the pipe. Returns
```

Sends '1 + 1' to the basic calculator and captures its output:

```
>>> subprocess.run('bc', input='1 + 1\n', capture_output=True,
CompletedProcess(args='bc', returncode=0, stdout='2\n', stderr
```

Sends test.in to the basic calculator running in standard mode and saves its output to test.out:

```
>>> from shlex import split
>>> os.popen('echo 1 + 1 > test.in')
>>> subprocess.run(split('bc -s'), stdin=open('test.in'), stdc
CompletedProcess(args=['bc', '-s'], returncode=0)
>>> open('test.out').read()
'2\n'
```

JSON

Text file format for storing collections of strings and numbers.

```
import json
<str>    = json.dumps(<object>)     # Converts object to JSON
<object> = json.loads(<str>)        # Converts JSON string to
```

Read Object from JSON File

```
def read_json_file(filename):
    with open(filename, encoding='utf-8') as file:
        return json.load(file)
```

Write Object to JSON File

```
def write_to_json_file(filename, an_object):
    with open(filename, 'w', encoding='utf-8') as file:
        json.dump(an_object, file, ensure_ascii=False, indent=
```

Pickle

Binary file format for storing Python objects.

```
import pickle
<bytes>  = pickle.dumps(<object>)   # Converts object to bytes
<object> = pickle.loads(<bytes>)    # Converts bytes object to
```

Read Object from File

```
def read_pickle_file(filename):
    with open(filename, 'rb') as file:
        return pickle.load(file)
```

Write Object to File

```
def write_to_pickle_file(filename, an_object):
    with open(filename, 'wb') as file:
        pickle.dump(an_object, file)
```

CSV

Text file format for storing spreadsheets.

```
import csv
```

Read

```
<reader> = csv.reader(<file>)       # Also: `dialect='excel',
<list>   = next(<reader>)           # Returns next row as a li
<list>   = list(<reader>)           # Returns a list of remair
```

- **File must be opened with a `'newline=""'` argument, or newlines embedded inside quoted fields will not be interpreted correctly!**
- **To print the spreadsheet to the console use Tabulate library.**
- **For XML and binary Excel files (xlsx, xlsm and xlsb) use Pandas library.**
- **Reader accepts any iterator of strings, not just files.**

Write

```
<writer> = csv.writer(<file>)       # Also: `dialect='excel',
<writer>.writerow(<collection>)     # Encodes objects using `s
<writer>.writerows(<coll_of_coll>)  # Appends multiple rows.
```

- File must be opened with a `'newline=""'` argument, or '\r' will be added in front of every '\n' on platforms that use '\r\n' line endings!
- Open existing file with `'mode="w"'` to overwrite it or `'mode="a"'` to append to it.

Parameters

- `'dialect'` - Master parameter that sets the default values. String or a 'csv.Dialect' object.
- `'delimiter'` - A one-character string used to separate fields.
- `'quotechar'` - Character for quoting fields that contain special characters.
- `'doublequote'` - Whether quotechars inside fields are/get doubled or escaped.
- `'skipinitialspace'` - Is space character at the start of the field stripped by the reader.
- `'lineterminator'` - How writer terminates rows. Reader is hardcoded to '\n', '\r', '\r\n'.
- `'quoting'` - 0: As necessary, 1: All, 2: All but numbers which are read as floats, 3: None.
- `'escapechar'` - Character for escaping quotechars if 'doublequote' is False.

Dialects

```
+------------------+--------------+--------------+---------
-----+
|                  |     excel    |   excel-tab  |     unix
|
+------------------+--------------+--------------+---------
-----+
| delimiter        |       ','    |      '\t'    |
','    |
| quotechar        |       '"'    |       '"'    |
'"'    |
| doublequote      |      True    |      True    |
True    |
| skipinitialspace |     False    |     False    |
False    |
| lineterminator   |    '\r\n'    |    '\r\n'    |
'\n'    |
| quoting          |         0    |         0    |
1    |
| escapechar       |      None    |      None    |
None    |
+------------------+--------------+--------------+---------
-----+
```

Read Rows from CSV File

```
def read_csv_file(filename, dialect='excel', **params):
    with open(filename, encoding='utf-8', newline='') as file:
        return list(csv.reader(file, dialect, **params))
```

Write Rows to CSV File

```
def write_to_csv_file(filename, rows, mode='w', dialect='excel
    with open(filename, mode, encoding='utf-8', newline='') as
        writer = csv.writer(file, dialect, **params)
        writer.writerows(rows)
```

SQLite

A server-less database engine that stores each database into a separate file.

```
import sqlite3
<conn> = sqlite3.connect(<path>)                # Opens existi
<conn>.close()                                  # Closes the c
```

Read

```
<cursor> = <conn>.execute('<query>')            # Can raise a
<tuple>  = <cursor>.fetchone()                  # Returns next
<list>   = <cursor>.fetchall()                  # Returns rema
```

Write

```
<conn>.execute('<query>')                       # Can raise a
<conn>.commit()                                 # Saves all ch
<conn>.rollback()                               # Discards all
```

Or:

```
with <conn>:                                    # Exits the bl
    <conn>.execute('<query>')                   # depending on
```

Placeholders

```
<conn>.execute('<query>', <list/tuple>)         # Replaces '?'
<conn>.execute('<query>', <dict/namedtuple>)    # Replaces ':<
<conn>.executemany('<query>', <coll_of_above>)  # Runs execute
```

- Passed values can be of type str, int, float, bytes, None, bool, datetime.date or datetime.datetime.
- Bools will be stored and returned as ints and dates as [ISO formatted

strings.

Example

Values are not actually saved in this example because `'conn.commit()'` is omitted!

```
>>> conn = sqlite3.connect('test.db')
>>> conn.execute('CREATE TABLE person (person_id INTEGER PRIM/
>>> conn.execute('INSERT INTO person VALUES (NULL, ?, ?)', ('J
1
>>> conn.execute('SELECT * FROM person').fetchall()
[(1, 'Jean-Luc', 187)]
```

SqlAlchemy

```
# $ pip3 install sqlalchemy
from sqlalchemy import create_engine, text
<engine> = create_engine('<url>')                # Url: 'dialec
<conn>   = <engine>.connect()                    # Creates a cc
<cursor> = <conn>.execute(text('<query>'), …)    # Replaces ':<
with <conn>.begin(): ...                         # Exits the b]
```

```
+------------+--------------+----------+------------------
---------------+
| Dialect    | pip3 install | import   |
Dependencies           |
+------------+--------------+----------+------------------
---------------+
| mysql      | mysqlclient  | MySQLdb  |
www.pypi.org/project/mysqlclient |
| postgresql | psycopg2     | psycopg2 |
www.pypi.org/project/psycopg2    |
| mssql      | pyodbc       | pyodbc   |
www.pypi.org/project/pyodbc      |
| oracle     | oracledb     | oracledb |
www.pypi.org/project/oracledb    |
+------------+--------------+----------+------------------
---------------+
```

Bytes

Bytes object is an immutable sequence of single bytes. Mutable version is called bytearray.

```
<bytes> = b'<str>'                          # Only accepts ASC
<int>   = <bytes>[<index>]                  # Returns an int i
<bytes> = <bytes>[<slice>]                  # Returns bytes ev
<bytes> = <bytes>.join(<coll_of_bytes>)     # Joins elements u
```

Encode

```
<bytes> = bytes(<coll_of_ints>)             # Ints must be in
<bytes> = bytes(<str>, 'utf-8')             # Or: <str>.encode
<bytes> = <int>.to_bytes(n_bytes, …)        # `byteorder='big/
<bytes> = bytes.fromhex('<hex>')            # Hex pairs can be
```

Decode

```
<list>  = list(<bytes>)                    # Returns ints in
<str>   = str(<bytes>, 'utf-8')            # Or: <bytes>.decc
<int>   = int.from_bytes(<bytes>, …)       # `byteorder='big/
'<hex>' = <bytes>.hex()                    # Returns hex pai1
```

Read Bytes from File

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

Write Bytes to File

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

Struct

- **Module that performs conversions between a sequence of numbers and a bytes object.**
- **System's type sizes, byte order, and alignment rules are used by default.**

```
from struct import pack, unpack
<bytes> = pack('<format>', <el_1> [, ...])  # Packages argumer
<tuple> = unpack('<format>', <bytes>)       # Use iter_unpack(
```

```
>>> pack('>hhl', 1, 2, 3)
b'\x00\x01\x00\x02\x00\x00\x00\x03'
>>> unpack('>hhl', b'\x00\x01\x00\x02\x00\x00\x00\x03')
(1, 2, 3)
```

Format

For standard type sizes and manual alignment (padding) start format string with:

- `'='` - System's byte order (usually little-endian).
- `'<'` - Little-endian.
- `'>'` - Big-endian (also `'!'`).

Besides numbers, pack() and unpack() also support bytes objects as part of the sequence:

- `'c'` - A bytes object with a single element. For pad byte use `'x'`.
- `'<n>s'` - A bytes object with n elements.

Integer types. Use a capital letter for unsigned type. Minimum and standard sizes are in brackets:

- `'b'` - char (1/1)
- `'h'` - short (2/2)
- `'i'` - int (2/4)
- `'l'` - long (4/4)
- `'q'` - long long (8/8)

Floating point types (struct always uses standard sizes):

- `'f'` - float (4/4)
- `'d'` - double (8/8)

Array

List that can only hold numbers of a predefined type. Available types and their minimum sizes in bytes are listed above. Type sizes and byte order are always determined by the system, however bytes of each element can be swapped with byteswap() method.

```
from array import array
<array> = array('<typecode>', <collection>)    # Array from cc
<array> = array('<typecode>', <bytes>)         # Array from by
<array> = array('<typecode>', <array>)         # Treats array
<array>.fromfile(<file>, n_items)              # Appends items
<bytes> = bytes(<array>)                       # Or: <array>.t
<file>.write(<array>)                          # Writes array
```

Memory View

- **A sequence object that points to the memory of another bytes-like object.**
- **Each element can reference a single or multiple consecutive bytes, depending on format.**
- **Order and number of elements can be changed with slicing.**
- **Casting only works between char and other types and uses system's sizes.**
- **Byte order is always determined by the system.**

```
<mview> = memoryview(<bytes/bytearray/array>)  # Immutable if
<real>  = <mview>[<index>]                     # Returns an ir
<mview> = <mview>[<slice>]                     # Mview with re
<mview> = <mview>.cast('<typecode>')           # Casts memory\
<mview>.release()                              # Releases the
```

```
<bytes> = bytes(<mview>)                       # Returns a nev
<bytes> = <bytes>.join(<coll_of_mviews>)       # Joins mviews
<array> = array('<typecode>', <mview>)         # Treats mview
<file>.write(<mview>)                          # Writes mview
```

```
<list>  = list(<mview>)                        # Returns a lis
<str>   = str(<mview>, 'utf-8')                # Treats mview
<int>   = int.from_bytes(<mview>, …)           # `byteorder='b
'<hex>' = <mview>.hex()                        # Treats mview
```

Deque

A thread-safe list with efficient appends and pops from either side. Pronounced "deck".

```
from collections import deque
<deque> = deque(<collection>)                    # Also `maxlen=
<deque>.appendleft(<el>)                         # Opposite elem
<deque>.extendleft(<collection>)                 # Collection ge
<el> = <deque>.popleft()                         # Raises IndexE
<deque>.rotate(n=1)                              # Rotates eleme
```

Threading

CPython interpreter can only run a single thread at a time. Using multiple threads won't result in a faster execution, unless at least one of the threads contains an I/O operation.

```
from threading import Thread, Timer, RLock, Semaphore, Event,
from concurrent.futures import ThreadPoolExecutor, as_complete
```

Thread

```
<Thread> = Thread(target=<function>)             # Use `args=<co
<Thread>.start()                                 # Starts the th
<bool> = <Thread>.is_alive()                     # Checks if the
<Thread>.join()                                  # Waits for the
```

- Use `'kwargs=<dict>'` to pass keyword arguments to the function.
- Use `'daemon=True'`, or the program will not be able to exit while the thread is alive.
- To delay thread execution use `'Timer(seconds, <func>)'` instead of Thread().

Lock

```
<lock> = RLock()                                # Lock that car
<lock>.acquire()                                # Waits for the
<lock>.release()                                # Makes the loc
```

Or:

```
with <lock>:                                    # Enters the bl
    ...                                         # exits it with
```

Semaphore, Event, Barrier

```
<Semaphore> = Semaphore(value=1)                # Lock that car
<Event>     = Event()                           # Method wait()
<Barrier>   = Barrier(n_times)                  # Wait() blocks
```

Queue

```
<Queue> = queue.Queue(maxsize=0)                # A thread-safe
<Queue>.put(<el>)                               # Blocks until
<Queue>.put_nowait(<el>)                        # Raises queue.
<el> = <Queue>.get()                            # Blocks until
<el> = <Queue>.get_nowait()                     # Raises queue.
```

Thread Pool Executor

```
<Exec> = ThreadPoolExecutor(max_workers=None)   # Or: `with Thr
<iter> = <Exec>.map(<func>, <args_1>, ...)      # Multithreaded
<Futr> = <Exec>.submit(<func>, <arg_1>, ...)    # Creates a thr
<Exec>.shutdown()                               # Blocks until
```

```
<bool> = <Future>.done()                        # Checks if the
<obj>  = <Future>.result(timeout=None)          # Waits for thı
<bool> = <Future>.cancel()                      # Cancels or re
<iter> = as_completed(<coll_of_Futures>)        # Next() waits
```

- Map() and as_completed() also accept 'timeout' argument. It causes TimeoutError when next() is called if result isn't available in 'timeout' seconds from the original call.
- Exceptions that happen inside threads are raised when next() is called on map's iterator or when result() is called on a Future. Its exception() method returns exception or None.
- ProcessPoolExecutor provides true parallelism, but everything sent to/from workers must be pickable. Queues must be sent using executor's 'initargs' and 'initializer' parameters.

Operator

Module of functions that provide the functionality of operators. Functions are ordered by operator precedence, starting with least binding.

```
import operator as op
<bool> = op.not_(<obj>)
<bool> = op.eq/ne/lt/le/gt/ge/contains/is_(<obj>, <obj>)
<obj>  = op.or_/xor/and_(<int/set>, <int/set>)
<int>  = op.lshift/rshift(<int>, <int>)
<obj>  = op.add/sub/mul/truediv/floordiv/mod(<obj>, <obj>)
<num>  = op.neg/invert(<num>)
<num>  = op.pow(<num>, <num>)
<func> = op.itemgetter/attrgetter/methodcaller(<obj> [, ...])
```

```
elementwise_sum  = map(op.add, list_a, list_b)
sorted_by_second = sorted(<collection>, key=op.itemgetter(1))
sorted_by_both   = sorted(<collection>, key=op.itemgetter(1, 0
product_of_elems = functools.reduce(op.mul, <collection>)
first_element    = op.methodcaller('pop', 0)(<list>)
```

- Bitwise operators require objects to have or(), xor(), and(), lshift(), rshift() and invert() special methods, unlike logical operators that work on all types of objects.
- Also: `'<bool> = <bool> &|^ <bool>'` and `'<int> = <bool> &|^ <int>'`.

Match Statement

Executes the first block with matching pattern. Added in Python 3.10.

```
match <object/expression>:
    case <pattern> [if <condition>]:
        <code>
    ...
```

Patterns

```
<value_pattern> = 1/'abc'/True/None/math.pi          # Matches
<class_pattern> = <type>()                           # Matches
<wildcard_patt> = _                                  # Matches
<capture_patt>  = <name>                             # Matches
<or_pattern>    = <pattern> | <pattern> [| ...]      # Matches
<as_pattern>    = <pattern> as <name>                # Binds t
<sequence_patt> = [<pattern>, ...]                   # Matches
<mapping_patt>  = {<value_pattern>: <pattern>, ...}  # Matches
<class_pattern> = <type>(<attr_name>=<patt>, ...)    # Matches
```

- Sequence pattern can also be written as a tuple.
- Use `'*<name>'` and `'**<name>'` in sequence/mapping patterns to bind remaining items.
- Sequence pattern must match all items, while mapping pattern does not.
- Patterns can be surrounded with brackets to override precedence (`'|'` > `'as'` > `','`).
- Built-in types allow a single positional pattern that is matched against

the entire object.

- All names that are bound in the matching case, as well as variables initialized in its block, are visible after the match statement.

Example

```
>>> from pathlib import Path
>>> match Path('/home/gto/python-cheatsheet/README.md'):
...     case Path(
...         parts=['/', 'home', user, *_],
...         stem=stem,
...         suffix=('.md' | '.txt') as suffix
...     ) if stem.lower() == 'readme':
...         print(f'{stem}{suffix} is a readme file that belor
'README.md is a readme file that belongs to user gto.'
```

Logging

```
import logging
```

```
logging.basicConfig(filename=<path>, level='DEBUG')  # Configu
logging.debug/info/warning/error/critical(<str>)     # Logs to
<Logger> = logging.getLogger(__name__)               # Logger
<Logger>.<level>(<str>)                              # Logs to
<Logger>.exception(<str>)                            # Calls e
```

Setup

```
logging.basicConfig(
    filename=None,                                   # Logs to
    format='%(levelname)s:%(name)s:%(message)s',     # Add '%(
    level=logging.WARNING,                           # Drops m
    handlers=[logging.StreamHandler(sys.stderr)]     # Uses Fi
)
```

```
<Formatter> = logging.Formatter('<format>')          # Creates
<Handler> = logging.FileHandler(<path>, mode='a')    # Creates
<Handler>.setFormatter(<Formatter>)                  # Adds Fo
<Handler>.setLevel(<int/str>)                        # Process
<Logger>.addHandler(<Handler>)                       # Adds Ha
<Logger>.setLevel(<int/str>)                         # What is
```

- Parent logger can be specified by naming the child logger `'<parent>.<name>'`.
- If logger doesn't have a set level it inherits it from the first ancestor that does.
- Formatter also accepts: pathname, filename, funcName, lineno, thread and process.
- A `'handlers.RotatingFileHandler'` creates and deletes log files based on 'maxBytes' and 'backupCount' arguments.

Creates a logger that writes all messages to file and sends them to the root's handler that prints warnings or higher:

```
>>> logger = logging.getLogger('my_module')
>>> handler = logging.FileHandler('test.log', encoding='utf-8'
>>> formatter = logging.Formatter('%(asctime)s %(levelname)s:%
>>> handler.setFormatter(formatter)
>>> logger.addHandler(handler)
>>> logging.basicConfig(level='DEBUG')
>>> logging.root.handlers[0].setLevel('WARNING')
>>> logger.critical('Running out of disk space.')
CRITICAL:my_module:Running out of disk space.
>>> print(open('test.log').read())
2023-02-07 23:21:01,430 CRITICAL:my_module:Running out of disk
```

Introspection

```
<list> = dir()                              # Names of local va
<dict> = vars()                             # Dict of local var
<dict> = globals()                          # Dict of global va
```

```
<list> = dir(<object>)                      # Names of object's
<dict> = vars(<object>)                     # Dict of writable
<bool> = hasattr(<object>, '<attr_name>')   # Checks if getatt
value  = getattr(<object>, '<attr_name>')   # Raises AttributeE
setattr(<object>, '<attr_name>', value)     # Only works on obj
delattr(<object>, '<attr_name>')            # Same. Also `del <
```

```
<Sig>  = inspect.signature(<function>)      # Function's Signat
<dict> = <Sig>.parameters                   # Dict of Parameter
<memb> = <Param>.kind                       # Member of Paramet
<obj>  = <Param>.default                    # Default value or
<type> = <Param>.annotation                 # Type or Parameter
```

Coroutines

- Coroutines have a lot in common with threads, but unlike threads, they only give up control when they call another coroutine and they don't use as much memory.
- Coroutine definition starts with `'async'` and its call with `'await'`.
- `'asyncio.run(<coroutine>)'` is the main entry point for asynchronous programs.
- Functions wait(), gather() and as_completed() start multiple coroutines at the same time.
- Asyncio module also provides its own Queue, Event, Lock and Semaphore classes.

Runs a terminal game where you control an asterisk that must avoid numbers:

```
import asyncio, collections, curses, curses.textpad, enum, ran
```

```
P = collections.namedtuple('P', 'x y')          # Position
D = enum.Enum('D', 'n e s w')                   # Direction
W, H = 15, 7                                    # Width, Height

def main(screen):
    curses.curs_set(0)                          # Makes cursor
    screen.nodelay(True)                        # Makes getch()
    asyncio.run(main_coroutine(screen))         # Starts runnir

async def main_coroutine(screen):
    moves = asyncio.Queue()
    state = {'*': P(0, 0), **{id_: P(W//2, H//2) for id_ in ra
    ai    = [random_controller(id_, moves) for id_ in range(16
    mvc   = [human_controller(screen, moves), model(moves, sta
    tasks = [asyncio.create_task(cor) for cor in ai + mvc]
    await asyncio.wait(tasks, return_when=asyncio.FIRST_COMPLE

async def random_controller(id_, moves):
    while True:
        d = random.choice(list(D))
        moves.put_nowait((id_, d))
        await asyncio.sleep(random.triangular(0.01, 0.65))

async def human_controller(screen, moves):
    while True:
        key_mappings = {258: D.s, 259: D.n, 260: D.w, 261: D.e
        ch = screen.getch()
        if d := key_mappings.get(ch):
            moves.put_nowait(('*', d))
        await asyncio.sleep(0.005)

async def model(moves, state):
    while state['*'] not in (state[id_] for id_ in range(10)):
        id_, d = await moves.get()
        x, y   = state[id_]
        deltas = {D.n: P(0, -1), D.e: P(1, 0), D.s: P(0, 1), D
        dx, dy = deltas[d]
        state[id_] = P((x + dx) % W, (y + dy) % H)

async def view(state, screen):
    offset = P(curses.COLS//2 - W//2, curses.LINES//2 - H//2)
    while True:
        screen.erase()
```

```
        curses.textpad.rectangle(screen, offset.y-1, offset.x-
        for id_, p in state.items():
            screen.addstr(
                offset.y + (p.y - state['*'].y + H//2) % H,
                offset.x + (p.x - state['*'].x + W//2) % W,
                str(id_)
            )
        screen.refresh()
        await asyncio.sleep(0.005)

if __name__ == '__main__':
    start_time = time.perf_counter()
    curses.wrapper(main)
    print(f'You survived {time.perf_counter() - start_time:.21
```

Libraries

Progress Bar

```
# $ pip3 install tqdm
>>> import tqdm, time
>>> for el in tqdm.tqdm([1, 2, 3], desc='Processing'):
...     time.sleep(1)
Processing: 100%|██████████████████████| 3/3 [00:03<00:00,  1.06
```

Plot

```
# $ pip3 install matplotlib
import matplotlib.pyplot as plt

plt.plot/bar/scatter(x_data, y_data [, label=<str>])  # Or: pl
plt.legend()                                          # Adds a
plt.savefig(<path>)                                   # Saves
plt.show()                                            # Displa
plt.clf()                                             # Clears
```

Table

Prints a CSV file as an ASCII table:

```
# $ pip3 install tabulate
import csv, tabulate
with open('test.csv', encoding='utf-8', newline='') as file:
    rows = list(csv.reader(file))
print(tabulate.tabulate(rows, headers='firstrow'))
```

Curses

Runs a basic file explorer in the console:

```
# $ pip3 install windows-curses
import curses, os
from curses import A_REVERSE, KEY_DOWN, KEY_UP, KEY_LEFT, KEY_

def main(screen):
    ch, first, selected, paths = 0, 0, 0, os.listdir()
    while ch != ord('q'):
        height, width = screen.getmaxyx()
        screen.erase()
        for y, filename in enumerate(paths[first : first+heigh
            color = A_REVERSE if filename == paths[selected] e
            screen.addnstr(y, 0, filename, width-1, color)
        ch = screen.getch()
        selected += (ch == KEY_DOWN) - (ch == KEY_UP)
        selected = max(0, min(len(paths)-1, selected))
        first += (selected >= first + height) - (selected < fi
        if ch in [KEY_LEFT, KEY_RIGHT, KEY_ENTER, ord('\n'), c
            new_dir = '..' if ch == KEY_LEFT else paths[select
            if os.path.isdir(new_dir):
                os.chdir(new_dir)
                first, selected, paths = 0, 0, os.listdir()

if __name__ == '__main__':
    curses.wrapper(main)
```

PySimpleGUI

A weight converter GUI application:

```
# $ pip3 install PySimpleGUI
import PySimpleGUI as sg

text_box = sg.Input(default_text='100', enable_events=True, ke
dropdown = sg.InputCombo(['g', 'kg', 't'], 'kg', readonly=True
label    = sg.Text('100 kg is 220.462 lbs.', key='-OUTPUT-')
button   = sg.Button('Close')
window   = sg.Window('Weight Converter', [[text_box, dropdown]

while True:
    event, values = window.read()
    if event in [sg.WIN_CLOSED, 'Close']:
        break
    try:
        value = float(values['-VALUE-'])
    except ValueError:
        continue
    unit = values['-UNIT-']
    factors = {'g': 0.001, 'kg': 1, 't': 1000}
    lbs = value * factors[unit] / 0.45359237
    window['-OUTPUT-'].update(value=f'{value} {unit} is {lbs:g
window.close()
```

Scraping

Scrapes Python's URL and logo from its Wikipedia page:

```
# $ pip3 install requests beautifulsoup4
import requests, bs4, os

response   = requests.get('https://en.wikipedia.org/wiki/Pythc
document   = bs4.BeautifulSoup(response.text, 'html.parser')
table      = document.find('table', class_='infobox vevent')
python_url = table.find('th', text='Website').next_sibling.a['
logo_url   = table.find('img')['src']
logo       = requests.get(f'https:{logo_url}').content
filename   = os.path.basename(logo_url)
with open(filename, 'wb') as file:
    file.write(logo)
print(f'{python_url}, file://{os.path.abspath(filename)}')
```

Selenium

Library for scraping websites with dynamic content.

```
# $ pip3 install selenium
from selenium import webdriver

<Drv> = webdriver.Chrome/Firefox/Safari/Edge()          # Opens
<Drv>.get('<url>')                                      # Also
<El> = <Drv/El>.find_element('css selector', '<css>')   # '<tag
<list> = <Drv/El>.find_elements('xpath', '<xpath>')     # '//<t
<str> = <El>.get_attribute/get_property(<str>)          # Also
<El>.click/clear()                                      # Also
```

XPath — also available in browser's console via `'$x(<xpath>)'`:

```
<xpath>     = //<element>[/ or // <element>]            # Chilc
<xpath>     = //<element>/following::<element>          # Next
<element>   = <tag><conditions><index>                  # `<tag
<condition> = [<sub_cond> [and/or <sub_cond>]]          # `and`
<sub_cond>  = @<attr>="<val>"                           # `.="<
<sub_cond>  = contains(@<attr>, "<val>")                # Is <v
<sub_cond>  = [//]<element>                             # Has n
```

Web

Flask is a micro web framework/server. If you just want to open a html file in a web browser use `'webbrowser.open(<path>)'` instead.

```
# $ pip3 install flask
import flask
```

```
app = flask.Flask(__name__)
app.run(host=None, port=None, debug=None)
```

- Starts the app at `'http://localhost:5000'`. Use `'host="0.0.0.0"'` to run externally.
- Install a WSGI server like Waitress and a HTTP server such as Nginx for better security.
- Debug mode restarts the app whenever script changes and displays errors in the browser.

Static Request

```
@app.route('/img/<path:filename>')
def serve_file(filename):
    return flask.send_from_directory('dirname/', filename)
```

Dynamic Request

```
@app.route('/<sport>')
def serve_html(sport):
    return flask.render_template_string('<h1>{{title}}</h1>',
```

- Use `'render_template(filename, <kwargs>)'` to render file located in templates dir.
- To return an error code use `'abort(<int>)'` and to redirect use

`'redirect(<url>)'`.

- `'request.args[<str>]'` returns parameter from the query string (URL part after '?').
- Use `'session[key] = value'` to store session data like username, etc.

REST Request

```
@app.post('/<sport>/odds')
def serve_json(sport):
    team = flask.request.form['team']
    return {'team': team, 'odds': [2.09, 3.74, 3.68]}
```

Starts the app in its own thread and queries its REST API:

```
# $ pip3 install requests
>>> import threading, requests
>>> threading.Thread(target=app.run, daemon=True).start()
>>> url = 'http://localhost:5000/football/odds'
>>> request_data = {'team': 'arsenal f.c.'}
>>> response = requests.post(url, data=request_data)
>>> response.json()
{'team': 'arsenal f.c.', 'odds': [2.09, 3.74, 3.68]}
```

Profiling

```
from time import perf_counter
start_time = perf_counter()
...
duration_in_seconds = perf_counter() - start_time
```

Timing a Snippet

```
>>> from timeit import timeit
>>> timeit('list(range(10000))', number=1000, globals=globals(
0.19373
```

Profiling by Line

```
$ pip3 install line_profiler
$ echo '@profile
def main():
    a = list(range(10000))
    b = set(range(10000))
main()' > test.py
$ kernprof -lv test.py
Line #      Hits         Time  Per Hit   % Time  Line
Contents
==============================================================
     1                                           @profile
     2                                           def
main():
     3         1        253.4    253.4     32.2      a =
list(range(10000))
     4         1        534.1    534.1     67.8      b =
set(range(10000))
```

Call and Flame Graphs

```
$ apt/brew install graphviz && pip3 install gprof2dot
snakeviz
$ tail --lines=4 test.py > test.py
$ python3 -m cProfile -o test.prof test.py
$ gprof2dot --format=pstats test.prof | dot -T png -o
test.png
$ xdg-open/open test.png
$ snakeviz test.prof
```

Sampling and Memory Profilers

```
+--------------+----------+------------+------------------
------------+------+
| pip3 install |   Type   |   Target   |          How to
run           | Live |
+--------------+----------+------------+------------------
------------+------+
| pyinstrument | Sampling |    CPU     | pyinstrument
test.py          | No   |
| py-spy       | Sampling |    CPU     | py-spy top --
python3 test.py | Yes  |
| scalene      | Sampling | CPU+Memory | scalene test.py
| No   |
| memray       | Tracing  |   Memory   | memray run --live
test.py     | Yes  |
+--------------+----------+------------+------------------
------------+------+
```

NumPy

Array manipulation mini-language. It can run up to one hundred times faster than the equivalent Python code. An even faster alternative that runs on a GPU is called CuPy.

```
# $ pip3 install numpy
import numpy as np
```

```
<array> = np.array(<list/list_of_lists/…>)              # Retu
<array> = np.zeros/ones/empty(<shape>)                  # Also
<array> = np.arange(from_inc, to_exc, ±step)            # Also
<array> = np.random.randint(from_inc, to_exc, <shape>)  # Also
```

```
<view>  = <array>.reshape(<shape>)                      # Also
<array> = <array>.flatten()                             # Also
<view>  = <array>.transpose()                           # Or:
```

```
<array> = np.copy/abs/sqrt/log/int64(<array>)           # Retu
<array> = <array>.sum/max/mean/argmax/all(axis)         # Pass
<array> = np.apply_along_axis(<func>, axis, <array>)    # Func
```

```
<array> = np.concatenate(<list_of_arrays>, axis=0)      # Link
<array> = np.row_stack/column_stack(<list_of_arrays>)   # Trea
<array> = np.tile/repeat(<array>, <int/list> [, axis])  # Tile
```

- **Shape is a tuple of dimension sizes. A 100x50 RGB image has shape (50, 100, 3).**
- **Axis is an index of the dimension that gets aggregated. Leftmost dimension has index 0. Summing the RGB image along axis 2 will return a greyscale image with shape (50, 100).**

Indexing

```
<el>       = <2d_array>[row_index, column_index]        # <3d_
<1d_view>  = <2d_array>[row_index]                      # <3d_
<1d_view>  = <2d_array>[:, column_index]                # <3d_
<2d_view>  = <2d_array>[rows_slice, columns_slice]      # <3d_
```

```
<2d_array> = <2d_array>[row_indexes]                    # <3d_
<2d_array> = <2d_array>[:, column_indexes]              # <3d_
<1d_array> = <2d_array>[row_indexes, column_indexes]    # <3d_
<1d_array> = <2d_array>[row_indexes, column_index]      # <3d_
```

```
<2d_bools> = <2d_array> > <el/1d/2d_array>              # 1d_a
<1d/2d_a>  = <2d_array>[<2d/1d_bools>]                  # 1d_b
```

- **Indexes should not be tuples because Python converts `'obj[i, j]'` to `'obj[(i, j)]'`!**
- **`':'` returns a slice of all dimension's indexes. Omitted dimensions default to `':'`.**

- Any value that is broadcastable to the indexed shape can be assigned to the selection.

Broadcasting

Set of rules by which NumPy functions operate on arrays of different sizes and/or dimensions.

```
left  = [[0.1], [0.6], [0.8]]                          # Shap
right = [ 0.1 ,  0.6 ,  0.8 ]                          # Shap
```

1. If array shapes differ in length, left-pad the shorter shape with ones:

```
left  = [[0.1], [0.6], [0.8]]                          # Shap
right = [[0.1 ,  0.6 ,  0.8]]                          # Shap
```

2. If any dimensions differ in size, expand the ones that have size 1 by duplicating their elements:

```
left  = [[0.1,  0.1,  0.1],                            # Shap
         [0.6,  0.6,  0.6],
         [0.8,  0.8,  0.8]]

right = [[0.1,  0.6,  0.8],                            # Shap
         [0.1,  0.6,  0.8],
         [0.1,  0.6,  0.8]]
```

Example

For each point returns index of its nearest point (`[0.1, 0.6, 0.8] => [1, 2, 1]`):

```
>>> points = np.array([0.1, 0.6, 0.8])
 [ 0.1,  0.6,  0.8]
>>> wrapped_points = points.reshape(3, 1)
[[ 0.1],
 [ 0.6],
 [ 0.8]]
>>> distances = wrapped_points - points
[[ 0. , -0.5, -0.7],
 [ 0.5,  0. , -0.2],
 [ 0.7,  0.2,  0. ]]
>>> distances = np.abs(distances)
[[ 0. ,  0.5,  0.7],
 [ 0.5,  0. ,  0.2],
 [ 0.7,  0.2,  0. ]]
>>> distances[range(3), range(3)] = np.inf
[[ inf,  0.5,  0.7],
 [ 0.5,  inf,  0.2],
 [ 0.7,  0.2,  inf]]
>>> distances.argmin(1)
[1, 2, 1]
```

Image

```
# $ pip3 install pillow
from PIL import Image
```

```
<Image> = Image.new('<mode>', (width, height))  # Also `color=
<Image> = Image.open(<path>)                    # Identifies t
<Image> = <Image>.convert('<mode>')             # Converts ima
<Image>.save(<path>)                            # Selects form
<Image>.show()                                  # Opens image
```

```
<int/tuple> = <Image>.getpixel((x, y))          # Returns pixe
<Image>.putpixel((x, y), <int/tuple>)           # Updates pixe
<ImagingCore> = <Image>.getdata()               # Returns a fl
<Image>.putdata(<list/ImagingCore>)             # Updates pixe
<Image>.paste(<Image>, (x, y))                  # Draws passec
```

```
<Image> = <Image>.filter(<Filter>)              # `<Filter> =
<Image> = <Enhance>.enhance(<float>)            # `<Enhance> =
```

```
<array> = np.array(<Image>)                     # Creates a 2c
<Image> = Image.fromarray(np.uint8(<array>))    # Use `<array>
```

Modes

- `'L'` - 8-bit pixels, greyscale.
- `'RGB'` - 3x8-bit pixels, true color.
- `'RGBA'` - 4x8-bit pixels, true color with transparency mask.
- `'HSV'` - 3x8-bit pixels, Hue, Saturation, Value color space.

Examples

Creates a PNG image of a rainbow gradient:

```
WIDTH, HEIGHT = 100, 100
n_pixels = WIDTH * HEIGHT
hues = (255 * i/n_pixels for i in range(n_pixels))
img = Image.new('HSV', (WIDTH, HEIGHT))
img.putdata([(int(h), 255, 255) for h in hues])
img.convert('RGB').save('test.png')
```

Adds noise to the PNG image and displays it:

```
from random import randint
add_noise = lambda value: max(0, min(255, value + randint(-20,
img = Image.open('test.png').convert('HSV')
img.putdata([(add_noise(h), s, v) for h, s, v in img.getdata(
img.show()
```

Image Draw

```
from PIL import ImageDraw
<ImageDraw> = ImageDraw.Draw(<Image>)           # Object for a
<ImageDraw>.point((x, y))                       # Draws a poir
<ImageDraw>.line((x1, y1, x2, y2 [, ...]))      # To get anti-
<ImageDraw>.arc((x1, y1, x2, y2), deg1, deg2)   # Always draws
<ImageDraw>.rectangle((x1, y1, x2, y2))         # To rotate us
<ImageDraw>.polygon((x1, y1, x2, y2, ...))      # Last point g
<ImageDraw>.ellipse((x1, y1, x2, y2))           # To rotate us
<ImageDraw>.text((x, y), text, font=<Font>)     # `<Font> = Im
```

- Use `'fill=<color>'` to set the primary color.
- Use `'width=<int>'` to set the width of lines or contours.
- Use `'outline=<color>'` to set the color of the contours.
- Color can be an int, tuple, `'#rrggbb[aa]'` string or a color name.

Animation

Creates a GIF of a bouncing ball:

```
# $ pip3 install imageio
from PIL import Image, ImageDraw
import imageio

WIDTH, HEIGHT, R = 126, 126, 10
frames = []
for velocity in range(1, 16):
    y = sum(range(velocity))
    frame = Image.new('L', (WIDTH, HEIGHT))
    draw = ImageDraw.Draw(frame)
    draw.ellipse((WIDTH/2-R, y, WIDTH/2+R, y+R*2), fill='white
    frames.append(frame)
frames += reversed(frames[1:-1])
imageio.mimsave('test.gif', frames, duration=0.03)
```

Audio

```
import wave
```

```
<Wave>   = wave.open('<path>', 'rb')   # Opens the WAV file.
<int>    = <Wave>.getframerate()       # Returns number of fra
<int>    = <Wave>.getnchannels()       # Returns number of san
<int>    = <Wave>.getsampwidth()       # Returns number of byt
<params> = <Wave>.getparams()          # Returns collection of
<bytes>  = <Wave>.readframes(nframes)  # Returns next n frames
```

```
<Wave> = wave.open('<path>', 'wb')     # Opens WAV file for wr
<Wave>.setframerate(<int>)             # Pass 44100 for CD, 48
<Wave>.setnchannels(<int>)             # Pass 1 for mono, 2 fo
<Wave>.setsampwidth(<int>)             # Pass 2 for CD, 3 for
<Wave>.setparams(<params>)             # Sets all parameters.
<Wave>.writeframes(<bytes>)            # Appends frames to the
```

- **Bytes object contains a sequence of frames, each consisting of one or more samples.**
- **In a stereo signal, the first sample of a frame belongs to the left**

channel.

- Each sample consists of one or more bytes that, when converted to an integer, indicate the displacement of a speaker membrane at a given moment.
- If sample width is one byte, then the integer should be encoded unsigned.
- For all other sizes, the integer should be encoded signed with little-endian byte order.

Sample Values

```
+-----------+-----------+------+-----------+
| sampwidth |    min    | zero |    max    |
+-----------+-----------+------+-----------+
|     1     |         0 |  128 |       255 |
|     2     |    -32768 |    0 |     32767 |
|     3     |  -8388608 |    0 |   8388607 |
+-----------+-----------+------+-----------+
```

Read Float Samples from WAV File

```
def read_wav_file(filename):
    def get_int(bytes_obj):
        an_int = int.from_bytes(bytes_obj, 'little', signed=(s
        return an_int - 128 * (sampwidth == 1)
    with wave.open(filename, 'rb') as file:
        sampwidth = file.getsampwidth()
        frames = file.readframes(-1)
    bytes_samples = (frames[i : i+sampwidth] for i in range(0,
    return [get_int(b) / pow(2, sampwidth * 8 - 1) for b in by
```

Write Float Samples to WAV File

```
def write_to_wav_file(filename, float_samples, nchannels=1, sa
    def get_bytes(a_float):
        a_float = max(-1, min(1 - 2e-16, a_float))
        a_float += sampwidth == 1
        a_float *= pow(2, sampwidth * 8 - 1)
        return int(a_float).to_bytes(sampwidth, 'little', sigr
    with wave.open(filename, 'wb') as file:
        file.setnchannels(nchannels)
        file.setsampwidth(sampwidth)
        file.setframerate(framerate)
        file.writeframes(b''.join(get_bytes(f) for f in float_
```

Examples

Saves a 440 Hz sine wave to a mono WAV file:

```
from math import pi, sin
samples_f = (sin(i * 2 * pi * 440 / 44100) for i in range(100_
write_to_wav_file('test.wav', samples_f)
```

Adds noise to the mono WAV file:

```
from random import random
add_noise = lambda value: value + (random() - 0.5) * 0.03
samples_f = (add_noise(f) for f in read_wav_file('test.wav'))
write_to_wav_file('test.wav', samples_f)
```

Plays the WAV file:

```
# $ pip3 install simpleaudio
from simpleaudio import play_buffer
with wave.open('test.wav', 'rb') as file:
    p = file.getparams()
    frames = file.readframes(-1)
    play_buffer(frames, p.nchannels, p.sampwidth, p.framerate)
```

Text to Speech

```
# $ pip3 install pyttsx3
import pyttsx3
engine = pyttsx3.init()
engine.say('Sally sells seashells by the seashore.')
engine.runAndWait()
```

Synthesizer

Plays Popcorn by Gershon Kingsley:

```
# $ pip3 install simpleaudio
import array, itertools as it, math, simpleaudio

F  = 44100
P1 = '71♩,69♪,,71♩,66♪,,62♩,66♪,,59♩,,,71♩,69♪,,71♩,66♪,,62♩,6
P2 = '71♩,73♪,,74♩,73♪,,74♪,,71♪,,73♩,71♪,,73♪,,69♪,,71♩,69♪,,
get_pause   = lambda seconds: it.repeat(0, int(seconds * F))
sin_f       = lambda i, hz: math.sin(i * 2 * math.pi * hz / F)
get_wave    = lambda hz, seconds: (sin_f(i, hz) for i in range
get_hz      = lambda note: 8.176 * 2 ** (int(note[:2]) / 12)
get_sec     = lambda note: 1/4 if '♩' in note else 1/8
get_samples = lambda note: get_wave(get_hz(note), get_sec(note
samples_f   = it.chain.from_iterable(get_samples(n) for n in (
samples_i   = array.array('h', (int(f * 30000) for f in sample
simpleaudio.play_buffer(samples_i, 1, 2, F).wait_done()
```

Pygame

```
# $ pip3 install pygame
import pygame as pg

pg.init()
screen = pg.display.set_mode((500, 500))
rect = pg.Rect(240, 240, 20, 20)
while not pg.event.get(pg.QUIT):
    deltas = {pg.K_UP: (0, -20), pg.K_RIGHT: (20, 0), pg.K_DOV
    for event in pg.event.get(pg.KEYDOWN):
        dx, dy = deltas.get(event.key, (0, 0))
        rect = rect.move((dx, dy))
    screen.fill((0, 0, 0))
    pg.draw.rect(screen, (255, 255, 255), rect)
    pg.display.flip()
```

Rectangle

Object for storing rectangular coordinates.

```
<Rect> = pg.Rect(x, y, width, height)          # Floats get t
<int>  = <Rect>.x/y/centerx/centery/…          # Top, right,
<tup.> = <Rect>.topleft/center/…               # Topright, bo
<Rect> = <Rect>.move((delta_x, delta_y))       # Use move_ip(
```

```
<bool> = <Rect>.collidepoint((x, y))           # Checks if re
<bool> = <Rect>.colliderect(<Rect>)            # Checks if tw
<int>  = <Rect>.collidelist(<list_of_Rect>)    # Returns inde
<list> = <Rect>.collidelistall(<list_of_Rect>) # Returns inde
```

Surface

Object for representing images.

Preview Code Blame Raw

```
<Surf> = pg.image.load(<path/file>)              # Loads the in
<Surf> = pg.surfarray.make_surface(<np_array>)   # Also `<np_ar
<Surf> = <Surf>.subsurface(<Rect>)               # Creates a ne
```

```
<Surf>.fill(color)                               # Tuple, Colo
<Surf>.set_at((x, y), color)                     # Updates pixe
<Surf>.blit(<Surf>, (x, y))                      # Draws passe
```

```
from pygame.transform import scale, ...
<Surf> = scale(<Surf>, (width, height))          # Returns scal
<Surf> = rotate(<Surf>, anticlock_degrees)       # Returns rota
<Surf> = flip(<Surf>, x_bool, y_bool)            # Returns flip
```

```
from pygame.draw import line, ...
line(<Surf>, color, (x1, y1), (x2, y2), width)   # Draws a line
arc(<Surf>, color, <Rect>, from_rad, to_rad)     # Also ellipse
rect(<Surf>, color, <Rect>, width=0)             # Also polygon
```

Font

```
<Font> = pg.font.Font(<path/file>, size)         # Loads TTF fi
<Surf> = <Font>.render(text, antialias, color)   # Background c
```

Sound

```
<Sound> = pg.mixer.Sound(<path/file/bytes>)      # WAV file or
<Sound>.play/stop()                              # Also <Sound>
```

Basic Mario Brothers Example

```
import collections, dataclasses, enum, io, itertools as it, p
```

```
from random import randint

P = collections.namedtuple('P', 'x y')          # Position
D = enum.Enum('D', 'n e s w')                   # Direction
W, H, MAX_S = 50, 50, P(5, 10)                  # Width, Heigh

def main():
    def get_screen():
        pg.init()
        return pg.display.set_mode((W*16, H*16))
    def get_images():
        url = 'https://gto76.github.io/python-cheatsheet/web/m
        img = pg.image.load(io.BytesIO(urllib.request.urlopen(
        return [img.subsurface(get_rect(x, 0)) for x in range(
    def get_mario():
        Mario = dataclasses.make_dataclass('Mario', 'rect spd
        return Mario(get_rect(1, 1), P(0, 0), False, it.cycle(
    def get_tiles():
        border = [(x, y) for x in range(W) for y in range(H) i
        platforms = [(randint(1, W-2), randint(2, H-2)) for _
        return [get_rect(x, y) for x, y in border + platforms]
    def get_rect(x, y):
        return pg.Rect(x*16, y*16, 16, 16)
    run(get_screen(), get_images(), get_mario(), get_tiles())

def run(screen, images, mario, tiles):
    clock = pg.time.Clock()
    pressed = set()
    while not pg.event.get(pg.QUIT) and clock.tick(28):
        keys = {pg.K_UP: D.n, pg.K_RIGHT: D.e, pg.K_DOWN: D.s,
        pressed |= {keys.get(e.key) for e in pg.event.get(pg.K
        pressed -= {keys.get(e.key) for e in pg.event.get(pg.K
        update_speed(mario, tiles, pressed)
        update_position(mario, tiles)
        draw(screen, images, mario, tiles, pressed)

def update_speed(mario, tiles, pressed):
    x, y = mario.spd
    x += 2 * ((D.e in pressed) - (D.w in pressed))
    x += (x < 0) - (x > 0)
    y += 1 if D.s not in get_boundaries(mario.rect, tiles) els
    mario.spd = P(x=max(-MAX_S.x, min(MAX_S.x, x)), y=max(-MAX

def update_position(mario, tiles):
```

```
    x, y = mario.rect.topleft
    n_steps = max(abs(s) for s in mario.spd)
    for _ in range(n_steps):
        mario.spd = stop_on_collision(mario.spd, get_boundarie
        mario.rect.topleft = x, y = x + (mario.spd.x / n_steps

def get_boundaries(rect, tiles):
    deltas = {D.n: P(0, -1), D.e: P(1, 0), D.s: P(0, 1), D.w:
    return {d for d, delta in deltas.items() if rect.move(delt

def stop_on_collision(spd, bounds):
    return P(x=0 if (D.w in bounds and spd.x < 0) or (D.e in b
             y=0 if (D.n in bounds and spd.y < 0) or (D.s in b

def draw(screen, images, mario, tiles, pressed):
    def get_marios_image_index():
        if D.s not in get_boundaries(mario.rect, tiles):
            return 4
        return next(mario.frame_cycle) if {D.w, D.e} & pressed
    screen.fill((85, 168, 255))
    mario.facing_left = (D.w in pressed) if {D.w, D.e} & press
    screen.blit(images[get_marios_image_index() + mario.facing
    for t in tiles:
        screen.blit(images[18 if t.x in [0, (W-1)*16] or t.y i
    pg.display.flip()

if __name__ == '__main__':
    main()
```

Pandas

```
# $ pip3 install pandas matplotlib
import pandas as pd, matplotlib.pyplot as plt
```

Series

Ordered dictionary with a name.

```
>>> pd.Series([1, 2], index=['x', 'y'], name='a')
x    1
y    2
Name: a, dtype: int64
```

```
<Sr> = pd.Series(<list>)                        # Assigns Range
<Sr> = pd.Series(<dict>)                        # Takes dictior
<Sr> = pd.Series(<dict/Series>, index=<list>)   # Only keeps it
```

```
<el> = <Sr>.loc[key]                            # Or: <Sr>.iloc
<Sr> = <Sr>.loc[keys]                           # Or: <Sr>.iloc
<Sr> = <Sr>.loc[from_key : to_key_inclusive]    # Or: <Sr>.iloc
```

```
<el> = <Sr>[key/index]                          # Or: <Sr>.key
<Sr> = <Sr>[keys/indexes]                       # Or: <Sr>[<key
<Sr> = <Sr>[bools]                              # Or: <Sr>.loc/
```

```
<Sr> = <Sr> > <el/Sr>                           # Returns a Ser
<Sr> = <Sr> + <el/Sr>                           # Items with no
```

```
<Sr> = pd.concat(<coll_of_Sr>)                  # Concats multi
<Sr> = <Sr>.combine_first(<Sr>)                 # Adds items th
<Sr>.update(<Sr>)                               # Updates items
```

```
<Sr>.plot.line/area/bar/pie/hist()              # Generates a M
plt.show()                                      # Displays the
```

Series — Aggregate, Transform, Map:

```
<el> = <Sr>.sum/max/mean/idxmax/all()          # Or: <Sr>.agg(
<Sr> = <Sr>.rank/diff/cumsum/ffill/interplt()  # Or: <Sr>.agg(
<Sr> = <Sr>.fillna(<el>)                       # Or: <Sr>.agg(
```

```
>>> sr = pd.Series([2, 3], index=['x', 'y'])
x    2
y    3
```

```
+---------------+-------------+-------------+--------------
-+
|               |    'sum'    |   ['sum']   | {'s': 'sum'}
|
+---------------+-------------+-------------+--------------
-+
| sr.apply(…)   |      5      |    sum  5   |     s  5
|
| sr.agg(…)     |             |             |
|
+---------------+-------------+-------------+--------------
-+
```

```
+---------------+-------------+-------------+--------------
-+
|               |    'rank'   |   ['rank']  | {'r': 'rank'}
|
+---------------+-------------+-------------+--------------
-+
| sr.apply(…)   |             |      rank   |
|
| sr.agg(…)     |     x  1    |   x     1   |    r  x  1
|
|               |     y  2    |   y     2   |       y  2
|
+---------------+-------------+-------------+--------------
-+
```

- **Keys/indexes/bools can't be tuples because `'obj[x, y]'` is converted**

to `'obj[(x, y)]'`!

- Methods ffill(), interpolate(), fillna() and dropna() accept `'inplace=True'`.
- Last result has a hierarchical index. Use `'<Sr>[key_1, key_2]'` to get its values.

DataFrame

Table with labeled rows and columns.

```
>>> pd.DataFrame([[1, 2], [3, 4]], index=['a', 'b'], columns=[
   x  y
a  1  2
b  3  4
```

```
<DF>    = pd.DataFrame(<list_of_rows>)         # Rows can be e
<DF>    = pd.DataFrame(<dict_of_columns>)      # Columns can b
```

```
<el>    = <DF>.loc[row_key, column_key]        # Or: <DF>.iloc
<Sr/DF> = <DF>.loc[row_key/s]                  # Or: <DF>.iloc
<Sr/DF> = <DF>.loc[:, column_key/s]            # Or: <DF>.iloc
<DF>    = <DF>.loc[row_bools, column_bools]    # Or: <DF>.iloc
```

```
<Sr/DF> = <DF>[column_key/s]                   # Or: <DF>.colu
<DF>    = <DF>[row_bools]                      # Keeps rows as
<DF>    = <DF>[<DF_of_bools>]                  # Assigns NaN t
```

```
<DF>    = <DF> > <el/Sr/DF>                    # Returns DF of
<DF>    = <DF> + <el/Sr/DF>                    # Items with no
```

```
<DF>    = <DF>.set_index(column_key)          # Replaces row
<DF>    = <DF>.reset_index(drop=False)        # Drops or move
<DF>    = <DF>.sort_index(ascending=True)     # Sorts rows by
<DF>    = <DF>.sort_values(column_key/s)      # Sorts rows by
```

DataFrame — Merge, Join, Concat:

```
>>> l = pd.DataFrame([[1, 2], [3, 4]], index=['a', 'b'], colum
   x  y
a  1  2
b  3  4
>>> r = pd.DataFrame([[4, 5], [6, 7]], index=['b', 'c'], colum
   y  z
b  4  5
c  6  7
```

```
+-----------------------+--------------+-----------+----
--------+-------------------------+
|                       |    'outer'   |   'inner' |
'left'   |       Description       |
+-----------------------+--------------+-----------+----
--------+-------------------------+
| l.merge(r, on='y',    |    x   y   z | x   y   z | x
y   z  | Merges on column if 'on' |
|           how=…)      | 0  1   2   . | 3   4   5 | 1
2   .  | or 'left/right_on' are   |
|                       | 1  3   4   5 |           | 3
4   5  | set, else on shared cols.|
|                       | 2  .   6   7 |           |
| Uses 'inner' by default. |
+-----------------------+--------------+-----------+----
--------+-------------------------+
| l.join(r, lsuffix='l', |    x yl yr  z |           | x
yl yr  z | Merges on row keys.      |
|          rsuffix='r', | a  1  2  .  . | x yl yr  z | 1
2  .  . | Uses 'left' by default.  |
|          how=…)       | b  3  4  4  5 | 3  4  4  5 | 3
4  4  5 | If r is a Series, it is  |
|                       | c  .  .  6  7 |           |
| treated as a column.     |
```

```
+-----------------------+--------------+-----------+----
--------+-------------------------+
| pd.concat([l, r],      |    x   y   z  |     y      |
| Adds rows at the bottom. |
|           axis=0,      | a  1   2   .  |     2      |
| Uses 'outer' by default. |
|           join=…)      | b  3   4   .  |     4      |
| A Series is treated as a |
|                        | b  .   4   5  |     4      |
| column. To add a row use |
|                        | c  .   6   7  |     6      |
| pd.concat([l, DF([sr])]).|
+-----------------------+--------------+-----------+----
--------+-------------------------+
| pd.concat([l, r],      |    x  y  y  z |            |
| Adds columns at the      |
|           axis=1,      | a  1  2  .  . | x  y  y  z |
| right end. Uses 'outer'  |
|           join=…)      | b  3  4  4  5 | 3  4  4  5 |
| by default. A Series is  |
|                        | c  .  .  6  7 |            |
| treated as a column.     |
+-----------------------+--------------+-----------+----
--------+-------------------------+
| l.combine_first(r)     |    x   y   z  |            |
| Adds missing rows and    |
|                        | a  1   2   .  |            |
| columns. Also updates    |
|                        | b  3   4   5  |            |
| items that contain NaN.  |
|                        | c  .   6   7  |            |
| Argument r must be a DF. |
+-----------------------+--------------+-----------+----
--------+-------------------------+
```

DataFrame — Aggregate, Transform, Map:

```
<Sr> = <DF>.sum/max/mean/idxmax/all()          # Or: <DF>.app]
<DF> = <DF>.rank/diff/cumsum/ffill/interplt()  # Or: <DF>.app]
<DF> = <DF>.fillna(<el>)                       # Or: <DF>.app]
```

- All operations operate on columns by default. Pass `'axis=1'` to process the rows instead.

```
>>> df = pd.DataFrame([[1, 2], [3, 4]], index=['a', 'b'], colu
   x  y
a  1  2
b  3  4
```

```
+----------------+------------+------------+-----------
---+
|                |    'sum'   |   ['sum']  | {'x':
'sum'}  |
+----------------+------------+------------+-----------
---+
| df.apply(…)    |     x  4   |       x  y |     x  4
|
| df.agg(…)      |     y  6   |  sum  4  6 |
|
+----------------+------------+------------+-----------
---+
```

```
+----------------+------------+------------+-----------
---+
|                |   'rank'   |  ['rank']  | {'x':
'rank'} |
+----------------+------------+------------+-----------
---+
| df.apply(…)    |            |      x   y |
|
| df.agg(…)      |      x  y  |   rank rank |        x
|
| df.transform(…) |   a  1  1  | a    1    1 |     a  1
|
|                |   b  2  2  | b    2    2 |     b  2
|
+----------------+------------+------------+-----------
---+
```

- Use `'<DF>[col_key_1, col_key_2][row_key]'` to get the fifth result's

values.

DataFrame — Plot, Encode, Decode:

```
<DF>.plot.line/area/bar/hist/scatter/box()      # Also: `x=colu
plt.show()                                      # Displays the
```

```
<DF> = pd.read_json/html('<str/path/url>')      # Run `$ pip3 i
<DF> = pd.read_csv('<path/url>')                # Also `names=<
<DF> = pd.read_pickle/excel('<path/url>')       # Use `sheet_na
<DF> = pd.read_sql('<table/query>', <conn.>)    # SQLite3/SQLA
```

```
<dict> = <DF>.to_dict(['d/l/s/…'])              # Returns colum
<str>  = <DF>.to_json/html/csv([<path>])        # Also to_marko
<DF>.to_pickle/excel(<path>)                    # Run `$ pip3 i
<DF>.to_sql('<table_name>', <connection>)       # Also `if_exis
```

GroupBy

Object that groups together rows of a dataframe based on the value of the passed column.

```
>>> df = pd.DataFrame([[1, 2, 3], [4, 5, 6], [7, 8, 6]], list(
>>> df.groupby('z').get_group(6)
   x  y  z
b  4  5  6
c  7  8  6
```

```
<GB> = <DF>.groupby(column_key/s)               # Splits DF int
<DF> = <GB>.apply(<func>)                       # Maps each gro
<GB> = <GB>[column_key]                         # Single column
<Sr> = <GB>.size()                              # A Sr of group
```

GroupBy — Aggregate, Transform, Map:

```
<DF> = <GB>.sum/max/mean/idxmax/all()          # Or: <GB>.agg(
<DF> = <GB>.rank/diff/cumsum/ffill()           # Or: <GB>.trar
<DF> = <GB>.fillna(<el>)                       # Or: <GB>.trar
```

```
>>> gb = df.groupby('z'); gb.apply(print)
   x  y  z
a  1  2  3
   x  y  z
b  4  5  6
c  7  8  6
```

```
+-----------------+-------------+-------------+------------
-+---------------+
|                 |    'sum'    |    'rank'   |   ['rank']
| {'x': 'rank'} |
+-----------------+-------------+-------------+------------
-+---------------+
| gb.agg(…)       |      x   y  |             |      x    y
|               |
|                 |  z          |      x  y   |   rank rank
|        x      |
|                 |  3   1   2  |   a  1  1   | a    1    1
|     a  1      |
|                 |  6  11  13  |   b  1  1   | b    1    1
|     b  1      |
|                 |             |   c  2  2   | c    2    2
|     c  2      |
+-----------------+-------------+-------------+------------
-+---------------+
| gb.transform(…) |      x   y  |      x  y   |
|               |
|                 |  a   1   2  |   a  1  1   |
|               |
|                 |  b  11  13  |   b  1  1   |
|               |
|                 |  c  11  13  |   c  2  2   |
|               |
+-----------------+-------------+-------------+------------
-+---------------+
```

Rolling

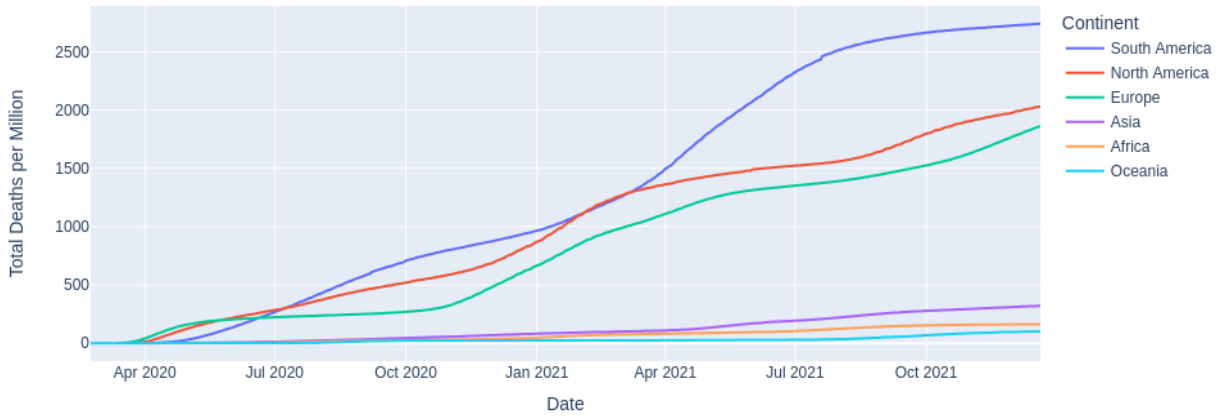
Object for rolling window calculations.

```
<RSr/RDF/RGB> = <Sr/DF/GB>.rolling(win_size)   # Also: `min_pe
<RSr/RDF/RGB> = <RDF/RGB>[column_key/s]        # Or: <RDF/RGB>
<Sr/DF>       = <R>.mean/sum/max()             # Or: <R>.apply
```

Plotly

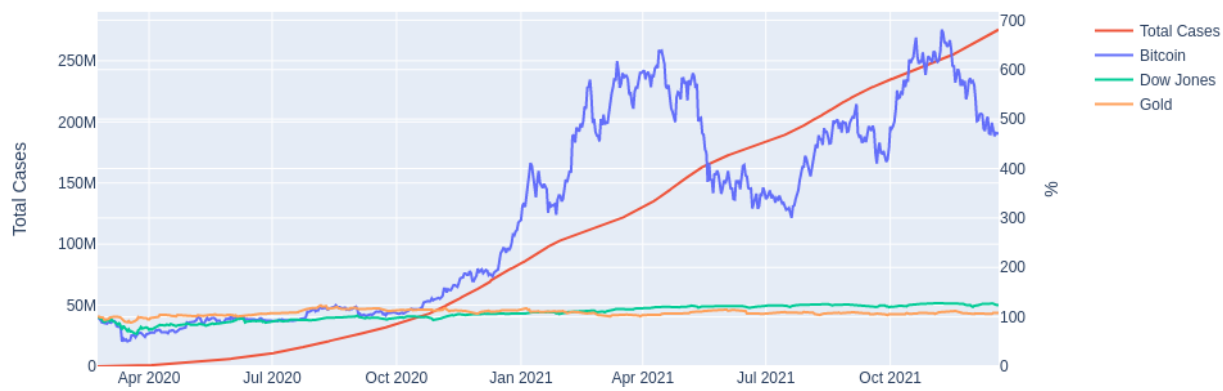
```
# $ pip3 install pandas plotly kaleido
import pandas as pd, plotly.express as ex
<Figure> = ex.line(<DF>, x=<col_name>, y=<col_name>)        #
<Figure>.update_layout(margin=dict(t=0, r=0, b=0, l=0), …)  #
<Figure>.write_html/json/image('<path>')                    #
```

Displays a line chart of total coronavirus deaths per million grouped by continent:



```
covid = pd.read_csv('https://covid.ourworldindata.org/data/owi
                    usecols=['iso_code', 'date', 'total_deaths
continents = pd.read_csv('https://gist.githubusercontent.com/s
                         '846ea5d35e5fc47f26c/raw/country-and-
                         usecols=['Three_Letter_Country_Code',
df = pd.merge(covid, continents, left_on='iso_code', right_on=
df = df.groupby(['Continent_Name', 'date']).sum().reset_index(
df['Total Deaths per Million'] = df.total_deaths * 1e6 / df.pc
df = df[df.date > '2020-03-14']
df = df.rename({'date': 'Date', 'Continent_Name': 'Continent']
ex.line(df, x='Date', y='Total Deaths per Million', color='Cor
```

Displays a multi-axis line chart of total coronavirus cases and changes in prices of Bitcoin, Dow Jones and gold:




```
import pandas as pd, plotly.graph_objects as go

def main():
    covid, bitcoin, gold, dow = scrape_data()
    display_data(wrangle_data(covid, bitcoin, gold, dow))

def scrape_data():
    def get_covid_cases():
        url = 'https://covid.ourworldindata.org/data/owid-covi
        df = pd.read_csv(url, usecols=['location', 'date', 'to
        return df[df.location == 'World'].set_index('date').to
    def get_ticker(symbol):
        url = (f'https://query1.finance.yahoo.com/v7/finance/c
               'period1=1579651200&period2=9999999999&interval
        df = pd.read_csv(url, usecols=['Date', 'Close'])
```

```
        return df.set_index('Date').Close
    out = get_covid_cases(), get_ticker('BTC-USD'), get_ticke
    return map(pd.Series.rename, out, ['Total Cases', 'Bitcoi

def wrangle_data(covid, bitcoin, gold, dow):
    df = pd.concat([bitcoin, gold, dow], axis=1)  # Creates ta
    df = df.sort_index().interpolate()            # Sorts tab
    df = df.loc['2020-02-23':]                    # Discards 
    df = (df / df.iloc[0]) * 100                  # Calculate
    df = df.join(covid)                           # Adds colu
    return df.sort_values(df.index[-1], axis=1)   # Sorts col

def display_data(df):
    figure = go.Figure()
    for col_name in reversed(df.columns):
        yaxis = 'y1' if col_name == 'Total Cases' else 'y2'
        trace = go.Scatter(x=df.index, y=df[col_name], name=c
        figure.add_trace(trace)
    figure.update_layout(
        yaxis1=dict(title='Total Cases', rangemode='tozero'),
        yaxis2=dict(title='%', rangemode='tozero', overlaying
        legend=dict(x=1.08),
        width=944,
        height=423
    )
    figure.show()

if __name__ == '__main__':
    main()
```

Appendix

Cython

Library that compiles Python code into C.

```
# $ pip3 install cython
import pyximport; pyximport.install()
import <cython_script>
<cython_script>.main()
```

Definitions:

- All `'cdef'` definitions are optional, but they contribute to the speed-up.
- Script needs to be saved with a `'pyx'` extension.

```
cdef <ctype> <var_name> = <el>
cdef <ctype>[n_elements] <var_name> = [<el>, <el>, ...]
cdef <ctype/void> <func_name>(<ctype> <arg_name>): ...
```

```
cdef class <class_name>:
    cdef public <ctype> <attr_name>
    def __init__(self, <ctype> <arg_name>):
        self.<attr_name> = <arg_name>
```

```
cdef enum <enum_name>: <member_name>, <member_name>, ...
```

Virtual Environments

System for installing libraries directly into project's directory.

```
$ python3 -m venv <name>       # Creates virtual environment ir
$ source <name>/bin/activate   # Activates venv. On Windows rur
$ pip3 install <library>       # Installs the library into acti
$ python3 <path>               # Runs the script in active envi
$ deactivate                   # Deactivates the active virtual
```

Basic Script Template

```python
#!/usr/bin/env python3
#
# Usage: .py
#

from sys import argv, exit
from collections import defaultdict, namedtuple
from dataclasses import make_dataclass
from enum import Enum
import functools as ft, itertools as it, operator as op, re


def main():
    pass


###
##  UTIL
#

def read_file(filename):
    with open(filename, encoding='utf-8') as file:
        return file.readlines()


if __name__ == '__main__':
    main()
```

Index

- Only available in the PDF.
- Ctrl+F / ⌘F is usually sufficient.
- Searching `'#<title>'` on the webpage will limit the search to the titles.