Cheatography

# Intermediate Python Cheat Sheet

by Mr Kitty via cheatography.com/23005/cs/4937/

## Built-in Functions

| | |
|---|---|
| float() | int() |
| bin(num) | hex(num) |
| dict() | list() |
| tuple() | str() |
| complex(a, b) | bool(x) |
| set() | sorted(s) |
| bytes(s) | bytearray(s) |
| abs(num) | len(s) |
| max(s) | min(s) |
| ord(char) | chr(num) |
| pow(x,y) | range([start] : stop : [step]) |
| round(num, places) | sum(s) |
| open(filename, [mode]) | type(obj) |
| id(obj) | divmod(num, divisor) |
| input(prompt) | print(s) |

## JSON Module

| | |
|---|---|
| dump(obj, fp, skipkeys=False, ensure_ascii=True, check_circular=True, allow_nan=True, cls=None, indent=None, separators=None, default=None, sort_keys=False, **kw) | Serialize obj as a JSON formatted stream to fp (a .write()-supporting file-like object) |
| dumps([same arguments as above, minus "fp"]) | Serialize obj to a JSON formatted str |
| load([same as dump]) | Deserialize fp (a .read()-supporting file-like object containing a JSON document) to a Python object |

## JSON Module (cont)

| | |
|---|---|
| loads(s, [same arguments as dumps]) | Deserialize s (a str instance containing a JSON document) to a Python object |

JSON functions have a lot of arguments, you'll only need to use "obj", "fp", and "s" about 99% of the time though

## Subprocess Module

| | |
|---|---|
| subprocess.run(args, *, stdin=None, input=None, stdout=None, stderr=None, shell=False, timeout=None, check=False) | The recommended approach to invoking subprocesses. This does not capture stdout or stderr by default. To do so, pass subprocess.PIPE to the appropriate arguments |
| subprocess.call(args, *, stdin=None, stdout=None, stderr=None, shell=False, timeout=None) | Run the command described by args. Wait for command to complete, then return the returncode attribute |
| subprocess.check_output(**) | Run command with arguments and return its output. Same as run(..., check=True, stdout=PIPE).stdout |

Note that "**" means to use the same arguments as above

## Time Module

| | |
|---|---|
| time.clock() | On Unix, return the current processor time as a floating point number expressed in seconds |
| time.sleep(secs) | Suspend execution of the calling thread for the given number of seconds |

## Datetime Module

| | |
|---|---|
| datetime.date() | An idealized date |
| datetime.time() | An idealized time |
| datetime.datetime(year, month, day, hour=0, minute=0, second=0, microsecond=0, tzinfo=None) | A combination of time and date |
| datetime.timedelta(days=0, seconds=0, microseconds=0, milliseconds=0, minutes=0, hours=0, weeks=0) | A time difference |
| datetime.today() | Return the current day |
| datetime.now(tz=None) | Return the current time and date |
| datetime.date() | Return the date portion of a datetime object |
| datetime.time() | Return the time portion of a datetime object |
| datetime.weekday() | Return the day of the week. Monday = 0 |
| .strftime(format string) | Format a datetime string. "%A, %d. %B %Y %I:%M%p" gives "Tuesday, 21. November 2006 04:30PM" |

## Random Module

| | |
|---|---|
| random.seed(a=None, version=2) | Initialize the random number generator |
| random.randrange([start,] stop[, step]) | Return a randomly selected element from range(start, stop, step) |
| random.randint(a, b) | Return a random integer N such that a <= N <= b |

## Random Module (cont)

| | |
|---|---|
| random.choice(seq) | Return a random element from the non-empty sequence seq |
| random.shuffle(x) | Shuffle the sequence x in place |
| random.sample(population, k) | Return a k length list of unique elements chosen from the population sequence or set |
| random.random() | Return the next random floating point number in the range [0.0, 1.0) |
| random.normalvariate(mu, sigma) | Normal distribution. mu is the mean, and sigma is the standard deviation |

Warning: the pseudo-random generators of this module should not be used for security purposes.

## Os Module

| | |
|---|---|
| os.uname | Return the operating system, release, version and machine as a tuple |
| os.chdir(path) | Change working directory |
| os.getcwd() | Returns the current working directory |
| os.listdir(path='.') | Return a list containing the names of the entries in the directory given by path |
| os.system(command) | Execute the command (a string) in a subshell. Replaced by the subprocess module |

## Regular Expressions Module

| | |
|---|---|
| compile(pattern, flags=0) | Compile a regular expression pattern into a regular expression object ("regex") |
| regex.search(string-[, pos[, endpos]]) | Scan through string looking for a location where this regular expression produces a match, and return a corresponding match object |
| regex.match(string[, pos[, endpos]]) | If zero or more characters at the beginning of string match this regular expression, return a corresponding match object |
| regex.fullmatch(string[, pos[, endpos]]) | If the whole string matches this regular expression, return a corresponding match object |
| match.group([group1, ...]) | Returns one or more subgroups of the match. Group "0" is the entire match |
| match.groups(default=None) | Return a tuple containing all the subgroups of the match |

## Smtplib Module

| | |
|---|---|
| SMTP(host='', port=0, local_hostname=None, [timeout, ]source_address=None) | A SMTP instance encapsulates an SMTP connection. For normal use, you should only require the initialization/connect, sendmail(), and quit() methods |

## Smtplib Module (cont)

| | |
|---|---|
| MTP.connect(host='localhost', port=0) | Connect to a host on a given port. The defaults are to connect to the local host at the standard SMTP port (25) |
| SMTP.helo(name='') | Identify yourself to the SMTP server using HELO |
| SMTP.login(user, password) | Log in on an SMTP server that requires authentication |
| SMTP.starttls(keyfile=None, certfile=None, context=None) | Put the SMTP connection in TLS (Transport Layer Security) mode. All SMTP commands that follow will be encrypted |
| SMTP.sendmail(from_addr, to_addrs, msg, mail_options=[], rcpt_options=[]) | Send mail |
| SMTP.quit() | Terminate the SMTP session and close the connection |

## Threading Module

| | |
|---|---|
| Thread(group=None, target=None, name=None, args=(), kwargs={}, *, daemon=None) | The main class of the this module. You use this to initialise a new thread |
| Thread.start() | Start the thread's activity |
| Thread.join(timeout=None) | Wait until the thread terminates |
| Thread.is_alive() | Return whether the thread is alive |

By **Mr Kitty**
cheatography.com/mr-kitty/

Published 19th August, 2015.
Last updated 19th August, 2015.

## Threading Module (cont)

| | |
|---|---|
| Lock() | The class implementing primitive lock objects. Once a thread has acquired a lock, subsequent attempts to acquire it block, until it is released; any thread may release it |
| Lock.acquire(blocking=True, timeout=-1) | Acquire a lock, blocking or non-blocking |
| Lock.release() | Release a lock. This can be called from any thread, not only the thread which has acquired the lock |
| Semaphore(value=1) | This class implements semaphore objects. A semaphore manages a counter representing the number of release() calls minus the number of acquire() calls, plus an initial value. The acquire() method blocks if necessary until it can return without making the counter negative |
| Semaphore.acquire(blocking=True, timeout=None) | cquire a semaphore |
| Semaphore.release() | Release a semaphore, incrementing the internal counter by one |

## Threading Module (cont)

| | |
|---|---|
| BoundedSemaphore(value=1) | Class implementing bounded semaphore objects. A bounded semaphore checks to make sure its current value doesn't exceed its initial value |
| Timer(interval, function, args=None, kwargs=None) | Create a timer that will run function with arguments args and keyword arguments kwargs, after interval seconds have passed |
| Timer.cancel() | Stop the timer, and cancel the execution of the timer's action |

## Argparse Module

| | |
|---|---|
| ArgumentParser(prog=None, usage=None, description=None, prefix_chars='-', argument_default=None, add_help=True) | Create a new ArgumentParser object. All parameters should be passed as keyword arguments |
| ArgumentParser.add_argument(name or flags... [, action][, nargs][, const][, default][, type][, choices][, required][, help][, metavar][, dest]) | Define how a single command-line argument should be parsed |
| ArgumentParser.parse_args(args=None, namespace=None) | Convert argument strings to objects and assign them as attributes of the namespace. Return the populated namespace |

## Argparse Module (cont)

| | |
|---|---|
| ArgumentParser.print_usage(file=None) | Print a brief description of how the ArgumentParser should be invoked on the command line |
| ArgumentParser.print_help(file=None) | Print a help message, including the program usage and information about the arguments registered with the ArgumentParser |

## Traceback Module

| | |
|---|---|
| print_tb(traceback, limit=None, file=None) | Print up to limit stack trace entries from traceback. If limit is omitted or None, all entries are printed |
| print_exception(type, value, traceback, limit=None, file=None, chain=True) | Print exception information and up to limit stack trace entries from traceback to file. Note that it prints the exception type and value after the stack trace |

You can get the traceback and other debugging info with:
exc_type, exc_value, exc_traceback = sys.exc_info()
(exc is short for "Exception")

By **Mr Kitty**
cheatography.com/mr-kitty/

Published 19th August, 2015.
Last updated 19th August, 2015.