Standard Libraries

cheat sheet

File and Filesystem Operations

Name	Purpose	When to use	Alternatives
os	Interact with operating system (files, processes, env vars)	For cross-platform file management, environment control	subprocess - for advanced process control pathlib - for modern file paths
shutil	High-level file operations (copy, move, delete)	When working with whole files/directories.	send2trash - for safer file deletion pathlib - for simpler operations
glob	Pattern-based file search using wildcards	Simple file search with patterns lile *.txt	pathlib.glob() ir preferred in modern Python
pathlib	Object-oriented file path manipulations	For readable, modern path operations	Replaces os.path, glob, and parts of shutil
tempfile	Secure temporary file/directory creation	Writing data temporarily (e.g., during processing)	None as secure and reliable in stdlib

Mathematics and Numbers

Name	Purpose	When to use	Alternatives
math	Basic math functions and constants	For direct arithmetic, trigonometry, etc	numpy - faster, array based computation
random	Generate pseudo-random values	Simulations, games, randomized selections	numpy.random - more powerful secrets - for cryptographic randomness
statistics	Simple statistical computation	Quick access to mean, median, etc	pandas, scipy.stats - more features, better for large datasets
decimal	Precise decimal arithmetic	Financial or high-precision math	mpmath - for even higher precision
heapq	Min-heap implementation	Priority queues, greedy algorithms	queue.PriorityQueue, sortedcontainers - more flexible
bisect	Binary search and insertion on sorted lists	Insert/search while maintaining sorted order	sortedcontainers - optimized, drop-in replacement

Data Structures and Algorithms

Name	Purpose	When to use	Alternatives
array	Memory-efficient fixed-type arrays	When list overhead is too large	numpy.array - more powerful bytearay - for bytes
collections	Advanced data structures like deque, Counter, defaultdict	When built-ins are not enough	blist, sortedcontainers - more performant variants
reprlib	Create short string representations	Logging/Debugging large or recursive objects	Custom __repr__, or pprint
pprint	Nicely formats nested data for readability	Printing complex structures	json.dumps(..., indent=2) - for JSON rich - for beautiful output
enum	Symbolic constant management	Replace magic numbers/string	intenum - special case typing.Literal - in type hints
functools	Functional programming tools	Caching, decorators, partials.	Native language features or toolz more FP tools
itertools	Efficient looping and iterator building	Lazy iteration, memory-efficient looping	more-itertools - adds even higher-level functions
operator	Functional equivalents to Python operators	When passing operators as functions	Usually none, useful inside map, sorted, etc.

Text Processing and String Handling

Name	Purpose	When to use	Alternatives
re	Regular expression matching and manipulation	Pattern-based search, validation, extraction	regex - third-party, supports Unicode and lookbehind better
string	Constants and simple string utilities (e.g., ascii_letters, Template)	When you need predefined character sets or basic substitution	f-strings, str.format() - are preferred in modern Python
textwrap	Format or swap text into lines	For console/terminal output formatting or emails	rich - for styled text output
unicodedata	Access Unicode properties and normalization	Unicode-aware string manipulation and cleanup	None as standard as this in stdlib. For deeper i18n use Babel
difflib	Compute differences between sequences (like files)	Implementing diff, spell-check, fuzzy match	python-Levenshtein - much faster fuzzywuzzy - for matching strings
stringprep	Prepare Unicode text for network protocols	Underlying layer for domain names, SASL auth.	Rarely used directly; replaced by idna in most libraries
html.parser	Parse HTML usin a built-in event-driven parser	Light HTML scraping or valdiation	BeautifulSoup - easier and more powerful lxml

Internet Protocols and Web

Name	Purpose	When to use	Alternatives
urlib.request	Handle HTTP requests natively	Simple file downloads or scraping	requests - much more user-friendly httpx - async support
smtplib	Sends emails using SMTP	Sending mail from Python apps or scripts	yagmail, secure-smtplib - adds SSL support
email	Construct and parse MIME email messages	Composing emails with attachments or HTML	yagmail - simplifies email composition
json	Read/write JSON data	Working with APIs, configs, or data interchange	ujson, orjson - faster parsing
xmlrpc.client	Call remote procedures over HTTP using XML-RPC	Legacy systems or simple RPC	requests + JSON-RPC - for modern RPC
xmlrpc.server	Expose functions as XML-RPC services	Quick setup for remote method exposure	Flask or FastAPI - more scalable, JSON-friendly

Dates and Time

Name	Purpose	When to use	Alternatives
datetime	Handle dates, times, and formatting	Most date/time operations (e.g., timestamps, durations)	arrow, pendulum - better timezones, friendlier APIs
timeit	Benchmark small bits of code	Precise timing of function or expression performance	pytest-benchmarch - for more robust performance testing

Data Formats and Serialization

Name	Purpose	When to use	Alternatives
csv	Read/write CSV files	Working with flat tabular data	pandas - more flexible and powerful pyexcel - multi-format support
sqlite3	Embedded relational database engine	Lightweight, file-based SQL storage	SQLAlchemy or TinyDB - NoSQL alternative
zipfile	Create and extract ZIP archives	Compress or unpack ZIP files	pyminizip - supports password protection
tarfile	Read/write TAR archives, including compressed ones	Backup/archive tasks	shutil.make_archive, libarchive - for more formats
zlib / bz2 / lzma	Compress and decompress data (DEFLATE, Bzip2, LZMA)	Binary compression and decompression	blosc, zstandard - faster and better compression rations

Security and Cryptography

Name	Purpose	When to use	Alternatives
hashlib	Generate hash digests (MD5, SHA-256, etc.)	File checksums, password hashing (non-secure), data integrity	bcrypy, argon2 - for secure hashing cryptography - for advanced crypto
hmac	Secure message digests using a shared secret	API authentication, token validation, secure comparisons	Built-in HMAC is still recommended. For tokens: itsdangerous
secrets	Generate cryptographically secure random numbers and tokens	Passwords, API keys, security tokens	None in stdlib - better than random for secure contexts

Utilities and Language Features

Name	Purpose	When to use	Alternatives
sys	Access system-level functionality like I/O, path, exit codes.	CLI tools, script termination, dynamic module control	Rarely replaced - core Python utility
argparse	Parse command-line arguments with help/validation	CLI applications needing robust arguments parsing	click - cleaner and more intuitive typer - modern CLI apps
contextlib	Utilities for writing custom context managers	Replace try-finally blocks, use with @contextmanager	closing(), ExitStacks() - are idiomatic; no real replacement
abc	Create abstract base classes and enforce implementation	Designing class hierarchies and enforcing interfaces	typing.Protocol - more flexible in static typing
typing	Type hints for static analysis and IDE support	Any large-scale or team project with type checking	None. Static typing in mypy, pyright - relies on it
dataclasses	Auto-generate boilerplate for data-handling classes	Simplify __init__, __repr__, comparisons	attrs - more features, validators, converters
warnings	Issue runtime warnings (e.g., deprecations)	Mark deprecated code, API changes.	No replacement - very standard.
importlib	Dynamically import and reload modules	Plugin systems, advanced meta-programming	Use with care, usually no need unless advanced use
locale	Format text/numbers using regional settings	Currency, dates, decimals formatting based on locale	babel - much better for i18n/l10n formatting
gettext	String translation for internalization	Translating applications into multiple languages	babel - for modern i18n workflows polib - for .po files
codecs	Encode/decode files in various encodings	When working with legacy encodings (e.g., Latin-1, UTF-16)	open(... encoding='utf-8') handles most use cases now

Concurrency and Parallelism

Name	Purpose	When to use	Alternatives
threading	Run code concurrently using threads	I/O-bound tasks, background jobs	concurrent.futures.ThreadPoolExecutor, avoid for CPU-bound tasks
concurrent.futures	Thread/process pool execution	Run functions in parallel threads or processes	joblib - for scientific workflows ray - for distributed workloads
multiprocessing	Run tasks in a separate processes (true parallelism)	CPU-bound tasks like data processing	joblib - easier syntax dask - for scalable workflows
asyncio	Asyncrhonous I/O using event loop coroutines	Network I/O, long-latency tasks, async APIs	trio - simpler model anyio - unified API over asyncio/trio)
queue	Thread-safe FILO/LIFO /Priority queues.	Mult-thread communication	multiprocessing.Queue - for inter-process janus - sync + async queue

Testing and Debugging

Name	Purpose	When to use	Alternative
unittest	Write structured test cases and suites	Format unit testing in project	pytest - more concise and powerful
doctest	Test code examples in docstrings	Lightweight testing with documentation	Inline pytest examples, doctest is unique
trace	Track code execution line-by-line	Code coverage or debugging execution path	coverage.py - more features and reporting
pdb	Console-based debugger	Interactive step-by-step debugging	ipdb - IPython version debugpy - for VS Code integration
inspect	Introspect functions, clases, modules	Runtime reflection, signature extraction, decorators	Still best-in-class for introspection
traceback	Format and print exception stack traces	Custom error handlers, logging	rich.traceback - for nicer console formatting
cProfile / profile	Profile Python function execution time	Finding bottlenecks in CPU-bound code.	line_profiler, pyinstrument - more visual
pstats	Sort/Analyze profiling results	Post-process output from cProfile/profile	snakeviz - visual output

Memory and Garbage Collection

Name	Purpose	When to use	Alternative
gc	Interface to garbage collector	Detect memory leaks, tune GC behavior	None - essential for memory debugging
weakref	Reference objects without preventing their collection	Caching, memory-sensitive references	None built-in. Useful with observer patterns, caches