Python 3.8+ Cheatsheet

https://winstonbrown.me

Shortcut / Concept	Description	Example
List Comprehensions	Create lists in one line, filtering or modifying elements.	`[x**2 for x in range(10) if x % 2 == 0]`
Dictionary Comprehensions	Generate dictionaries on the fly with custom keys and values.	`{x: x**2 for x in range(5)}`
Lambda Functions	Write small anonymous functions inline, useful for quick operations.	`sorted(data, key=lambda x: x['age'])`
F-strings	Embed expressions directly in strings for clean, efficient formatting.	`name = "Winston"; f"Hello, {name}!"`
Enumerate	Get index and value from an iterable in a loop.	`for i, value in enumerate(lst): print(i, value)`
Zip	Combine multiple lists into tuples, iterating over each element.	`list(zip(list1, list2))`
Unpacking	Unpack elements from lists, tuples, and dictionaries into variables easily.	`a, b, c = my_tuple`
***args & **kwargs**	Allow functions to take an arbitrary number of arguments and keyword arguments.	`def func(*args, **kwargs):`
Ternary Operator	Write simple conditional expressions in one line.	`"even" if x % 2 == 0 else "odd"`
Generators	Use `yield` to create memory-efficient generators instead of full lists.	`def gen(): yield x`
Map, Filter, Reduce	Functional programming tools for element-wise mapping, filtering, or reducing a sequence.	`map(func, iterable)`

Shortcut / Concept	Description	Example
Context Managers	Manage resources using `with` statements (e.g., file handling).	`with open("file.txt") as f:`
Named Tuples	Create lightweight, readable, and immutable tuple-like objects.	`from collections import namedtuple; Point = namedtuple('Point', 'x y')`
Counter	Count elements in a list or other iterable with ease.	`from collections import Counter; Counter(lst)`
Defaultdict	Use dictionaries with default values for missing keys.	`from collections import defaultdict; d = defaultdict(int)`
Datetime Manipulation	Work with dates and times effectively.	`from datetime import datetime; now = datetime.now()`
Pathlib	Modern and efficient way to work with paths and file systems.	`from pathlib import Path; Path("myfile.txt").exists()`
List Flattening	Quickly flatten nested lists with itertools.	`from itertools import chain; list(chain(*nested_list))`
Dealing with JSON	Parse JSON objects easily from strings or files.	`import json; data = json.loads(json_str)`
Exception Handling with else	Use `else` in try-except blocks to run code only if no exception occurred.	`try: ... except: ... else: ...`
Type Hinting	Annotate code with type hints for readability and IDE support.	`def func(x: int) -> int:`