🐍 Python Cheatsheet v3.12+

Complete Reference for Seasoned Developers

🔧 Variables & References

```
# Variables are references, not containers
x = [1, 2, 3]
y = x  # y references same object as x
y.append(4)  # modifies x too
print(x)  # [1, 2, 3, 4]

# Identity vs equality
x is y  # True (same object)
x == [1, 2, 3, 4]  # True (same value)

# Mutable default trap
def bad(a, list=[]):  # list created once
    list.append(a)
    return list

# Fix: use None as default
def good(a, list=None):
    if list is None:
        list = []
    list.append(a)
    return list
```

📦 Collections

```
# List operations
list = [1, 2, 3]
list.append(4)  # [1, 2, 3, 4]
[0, *list]  # [0, 1, 2, 3, 4] (unpacking)

# Dict operations
d = {'a': 1, 'b': 2}
d.get('c', 0)  # 0 (default if key missing)
{**d, 'c': 3}  # {'a':1, 'b':2, 'c':3} (merge)
d | {'c': 3}   # Same merge (Python 3.9+)

# Set operations
s1, s2 = {1, 2, 3}, {3, 4, 5}
s1 & s2  # {3} (intersection)
s1 | s2  # {1, 2, 3, 4, 5} (union)
s1 - s2  # {1, 2} (difference)
```

📁 File Operations

```
# Reading files
with open('file.txt', 'r') as f:
    content = f.read()  # Read entire file

# Line by line (memory efficient)
with open('file.txt', 'r') as f:
    for line in f:
        process(line.strip())

# Writing files
with open('file.txt', 'w') as f:
    f.write('Hello\n')
    f.writelines(['line1\n', 'line2\n'])

# JSON handling
import json
data = json.loads('{"key": "value"}')
json_str = json.dumps(data, indent=2)
```

🔄 Iteration Patterns

```
# Enumerate with index
for i, val in enumerate(items):
    print(f"{i}: {val}")

# Dict iteration
for k, v in my_dict.items():
    print(f"{k}: {v}")

# Multiple sequences
for x, y in zip(list1, list2):
    print(x, y)

# Flatten nested lists
from itertools import chain
list(chain.from_iterable([[1, 2], [3, 4]]))

# Group by property
from itertools import groupby
{k: list(g) for k, g in
 groupby(items, key=lambda x: x.prop)}
```

🧱 Comprehensions

```
# List comprehension
[x*2 for x in range(10) if x % 2 == 0]
# [0, 4, 8, 12, 16]

# Nested comprehension (matrix transpose)
matrix = [[1, 2, 3], [4, 5, 6]]
[[row[i] for row in matrix] for i in range(3)]
# [[1, 4], [2, 5], [3, 6]]

# Dict comprehension
{x: x**2 for x in range(5)}
# {0: 0, 1: 1, 2: 4, 3: 9, 4: 16}

# Set comprehension
{x % 3 for x in range(10)}  # {0, 1, 2}

# Generator expression (memory efficient)
sum(x**2 for x in range(1000000))
```

⚙️ Data Processing

```
# Sorting
sorted(items, key=lambda x: x.attr, reverse=True)

# Filtering
list(filter(lambda x: x > 0, items))

# Mapping/transformation
list(map(lambda x: x**2, items))

# Reduce
from functools import reduce
reduce(lambda acc, x: acc + x, items, 0)

# Counter for frequency analysis
from collections import Counter
counts = Counter(['a', 'b', 'a', 'c'])
most_common = counts.most_common(2)

# Named tuples (lightweight classes)
from collections import namedtuple
Point = namedtuple('Point', ['x', 'y'])
p = Point(1, 2)
```

⚙️ Functions & Lambdas

```
# Args & kwargs
def f(pos, /, pos_or_kw, *, kw_only):
    pass
f(1, 2, kw_only=3)  # valid call

# Type hints
def greet(name: str) -> str:
    return f"Hello {name}"

# Multiple return values (as tuple)
def minmax(items):
    return min(items), max(items)
lo, hi = minmax([1, 2, 3])

# Lambda functions
sort_key = lambda obj: obj.name
squares = list(map(lambda x: x**2, range(5)))

# Partial application
from functools import partial
add5 = partial(add, 5)  # add(5, x)
```

🔤 String Operations

```
# f-strings (Python 3.6+)
name, age = "Alice", 30
f"{name} is {age} years old"

# String methods
"  text  ".strip()  # "text"
",".join(["a", "b", "c"])  # "a,b,c"
"a,b,c".split(",")  # ["a", "b", "c"]
"hello".replace("l", "L")  # "heLLo"

# String properties
"abc".isalpha()  # True
"123".isdigit()  # True
"hello".startswith("he")  # True
"hello".endswith("lo")  # True
```

❌ Exception Handling

```
# Basic try/except
try:
    result = risky_operation()
except (TypeError, ValueError) as e:
    print(f"Error: {e}")
else:  # Runs if no exceptions
    print("Success!")
finally:  # Always runs
    cleanup()

# Custom exception
class MyError(Exception):
    pass

# Context managers
with open('file.txt') as f:
    data = f.read()
```

📦 Modules & Packages

```
# Importing modules
import math
from datetime import datetime, timedelta
import numpy as np  # Common convention

# Relative imports (in packages)
from . import sibling_module
from ..parent import parent_module

# Creating modules
if __name__ == "__main__":
    # Code runs when module is executed directly
    main()
else:
    # Code runs when imported
    setup()
```

🏗️ Classes & Methods

```
class Point:
    # Class variable (shared by all instances)
    instances = 0

    def __init__(self, x, y):
        self.x, self.y = x, y  # instance vars
        Point.instances += 1

    # Regular method (receives self)
    def distance(self):
        return (self.x**2 + self.y**2)**0.5

    # Class method (receives cls)
    @classmethod
    def from_tuple(cls, tup):
        return cls(*tup)

    # Static method (receives nothing)
    @staticmethod
    def origin_distance(x, y):
        return (x**2 + y**2)**0.5

    # Property (access like attribute)
    @property
    def magnitude(self):
        return self.distance()
```

🔮 Special Methods

```
# Operator overloading
__add__(self, other)      # self + other
__getitem__(self, key)    # self[key]
__contains__(self, item)  # item in self
__len__(self)             # len(self)
__call__(self, *args)     # self(*args)
__str__(self)             # str(self)
__repr__(self)            # repr(self)
__enter__/__exit__        # with statement
__iter__/__next__         # iteration
```

🚀 Modern Python Features

```
# Walrus operator (:=) - Python 3.8+
if (n := len(data)) > 10:
    print(f"Processing {n} items")

# f-strings with = (Python 3.8+)
x, y = 10, 20
print(f"{x=}, {y=}")  # x=10, y=20

# Pattern matching - Python 3.10+
match command.split():
    case ["quit"]:
        return "Exiting"
    case ["load", filename]:
        return f"Loading {filename}"
    case _:  # Default case
        return "Unknown command"

# Structural pattern matching - Python 3.10+
match value:
    case [a, b, *rest]:
        return a + b
    case {'key': value}:
        return value
    case _:
        return default
```

🔗 Quick Links: python.org • pypi.org • Python Docs • PEPs