🐍 Python Cheatsheet

Quick Reference for Seasoned Developers

Variables & References

```python
# Variables are references
x = [1, 2, 3]
y = x  # y references same object
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 (value)

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

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

Collections

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

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

# Set operations
s1, s2 = {1, 2, 3}, {3, 4, 5}
s1 & s2  # {3}
s1 | s2  # {1, 2, 3, 4, 5}
s1 - s2  # {1, 2}
```

File Operations

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

# Line by line (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(['a\n', 'b\n'])

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

Iteration Patterns

```python
# 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
flat = list(chain.from_iterable(
    [[1, 2], [3, 4]]
))

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

Comprehensions

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

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

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

# Set comprehension
mods = {x % 3 for x in range(10)}

# Generator (efficient)
total = sum(x**2 for x in range(1000000))
```

Data Processing

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

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

# Mapping
squares = list(map(lambda x: x**2, items))

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

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

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

Functions & Lambdas

```python
# Positional/keyword-only args
def f(pos, /, pos_or_kw, *, kw_only):
    pass
f(1, 2, kw_only=3)  # valid

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

# Multiple return values
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)
```

String Operations

```python
# f-strings (Python 3.6+)
name, age = "Alice", 30
msg = 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

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

# Custom exception
class MyError(Exception):
    pass

# Raising exceptions
raise MyError("Something wrong")

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

Modules & Packages

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

# Relative imports
from . import sibling_module
from ..parent import parent_module

# Module guard
if __name__ == "__main__":
    # Runs when executed directly
    main()
else:
    # Runs when imported
    setup()
```

Classes & Methods

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

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

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

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

    # Static method
    @staticmethod
    def origin_distance(x, y):
        return (x**2 + y**2)**0.5

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

Special Methods

```python
# Operator overloading
__add__(self, other)      # +
__sub__(self, other)      # -
__mul__(self, other)      # *
__getitem__(self, key)    # self[key]
__setitem__(self, k, v)   # self[k] = v
__contains__(self, item)  # in
__len__(self)             # len()
__call__(self, *args)     # self()
__str__(self)             # str()
__repr__(self)            # repr()
__enter__/__exit__        # with
__iter__/__next__         # iteration
```

Modern Python Features

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

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

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

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