

Python Cheat Sheet: Advanced

Ch. 44-68 | engineering, big data, ML and AI, expert level

Performance and profiling

Ch. 44

```
$ python -m cProfile -o out.prof app.py
# who is slow? [chk]
$ kernprof -l -v app.py # line by line [chk]
import timeit # measure, don't guess
tm = timeit.timeit
l = tm('99 in x', 'x = list(range(100))')
s = tm('99 in x', 'x = set(range(100))')
s < l # -> True
import tracemalloc # watch memory
tracemalloc.start()
x = [0] * 100_000
tracemalloc.get_traced_memory()[0] > 10**5
# -> True
```

Packaging

Ch. 45

```
# pyproject.toml (from Ch. 45) [ref]
[build-system]
requires = ["setuptools>=77"]
build-backend = "setuptools.build_meta"
[project]
name = "field-monitor-example"
version = "0.1.0"
[project.scripts]
field-monitor = "field_monitor.cli:main"
$ python -m build # build sdist + wheel [chk]
$ twine check dist/* # check metadata [chk]
```

Python 3.12 - 3.14

Ch. 47

```
# 3.12: generics + type
def first[T](xs: list[T]) -> T:
    return xs[0]
type Items[T] = list[T]
first([3, 1]) # -> 3
# 3.12: same quotes in f-string
j = ['Teak', 'Mahogany']
f"{', '.join(j)}" # -> 'Teak, Mahogany'
import copy, datetime # 3.13: copy.replace
copy.replace(datetime.date(2026, 1, 1), month=6)
# -> datetime.date(2026, 6, 1)
name = 'Teak' # 3.14: t-string
t = t'Tree {name}'
type(t).__name__, t.strings
# -> ('Template', ('Tree', ''))
class Node: # 3.14: forward ref, no quotes
    def __init__(self, nxt: Node | None):
        self.nxt = nxt
Node(Node(None)).nxt.nxt is None # -> True
# 3.14: sub-interpreters [chk]
from concurrent.futures import (
    InterpreterPoolExecutor)
```

Docker and CI (reference)

Ch. 48

```
# Dockerfile (from Ch. 48) [ref]
FROM python:3.14-slim
WORKDIR /app
USER appuser
EXPOSE 8000
$ docker build -t monitor .
# Docker is not installed here [ref]
$ docker run -p 8000:8000 monitor
# open localhost:8000 [ref]
on: # GitHub Actions (from Ch. 48) [ref]
  push:
  jobs:
    test:
      runs-on: ubuntu-latest
```

Security

Ch. 49

```
# safe random token
import hashlib, hmac, secrets
len(secrets.token_hex(16)) # -> 32
hashlib.sha256(b'abc').hexdigest()[0:12]
# data fingerprint -> 'ba7816bf8f01'
# passwords: script
h = dict(salt=b'salt', n=2**14, r=8, p=1)
k = hashlib.scrypt(b'secret', **k)
h2 = hashlib.scrypt(b'secret', **k)
hmac.compare_digest(h, h2) # -> True
# reject ../ (traversal)
from pathlib import Path
d = Path('data').resolve()
(d / '..').secret.txt'.resolve().is_relative_to(d)
# -> False
import html # stop HTML injection
html.escape('<script>') # -> '&lt;script&gt;'
$ pip-audit -r requirements.txt
# find vulnerable packages [chk]
$ bandit -r src # find risky patterns [chk]
```

Polars, DuckDB, Parquet

Ch. 50

```
import polars as pl # fast tables
d = pl.DataFrame(dict(k=list('aba'), v=[1, 2, 3]))
g = d.group_by('k').agg(pl.col('v').sum())
g.sort('k').to_dicts()
# -> [{'k': 'a', 'v': 4}, {'k': 'b', 'v': 2}]
d.lazy().filter(pl.col('v') > 1).collect().shape
# lazy plan -> (2, 2)
# Parquet: columnar, compressed
d.write_parquet('a.parquet')
pl.read_parquet('a.parquet').shape # -> (3, 2)
import duckdb # SQL on a file
q = "SELECT k, SUM(v) FROM 'a.parquet' GROUP BY k"
sorted(duckdb.sql(q).fetchall())
# -> [('a', 4), ('b', 2)]
```

STAC, COG, Dask, xarray

Ch. 51, 53

```
import pystac # one scene = one Item
from datetime import datetime, timezone
t = datetime(2026, 1, 1, tzinfo=timezone.utc)
i = pystac.Item('s2-1', None, None, t, {})
i.to_dict()['type'] # -> 'Feature'
from pystac_client import Client
# search a STAC catalogue [chk]
s.read(1, window=Window(2, 2, 3, 3))
# COG: read a window (needs a network) [ref]
# chunks + lazy compute
import dask.array as da
a = da.ones((1000, 1000), chunks=500)
float(a.sum().compute()) # -> 1000000.0
# labelled arrays
import numpy as np, xarray as xr
a = np.arange(6).reshape(2, 3)
x = xr.DataArray(a, dims=('t', 'k'))
x.isel(t=1).sum().item() # -> 12
```

scikit-learn and boosting

Ch. 54

```
# synthetic data
from sklearn.datasets import make_classification
X, y = make_classification(200, random_state=0)
# 5-fold cross-validation
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import cross_val_score
m = RandomForestClassifier(50, random_state=0)
s = cross_val_score(m, X, y, cv=5)
round(float(s.mean()), 2) # -> 0.93
int(m.fit(X, y).feature_importances_.argmax())
# top feature -> 14
import xgboost as xgb # boosting
g = xgb.XGBClassifier(n_estimators=20)
g.fit(X, y).score(X, y) > 0.9 # -> True
```

PyTorch and GPU

Ch. 55

```
# train: zero, loss, backward, step
import torch
x = torch.tensor([1.], [2.], [3.])
net = torch.nn.Linear(1, 1)
opt = torch.optim.SGD(net.parameters(), lr=.05)
for _ in range(200):
    opt.zero_grad()
    loss = ((net(x) - 2 * x) ** 2).mean()
    loss.backward()
    opt.step()
round(net.weight.item(), 1) # -> 1.9
d = 'cuda' if torch.cuda.is_available() else 'cpu'
# GPU if present
net.to(d)(x.to(d)).shape
# model + data on one device -> torch.Size([3, 1])
```

Computer vision: YOLO

Ch. 56

```
# IoU: box overlap
ar = lambda b: (b[2] - b[0]) * (b[3] - b[1])
def iou(a, b):
    w = min(a[2], b[2]) - max(a[0], b[0])
    h = min(a[3], b[3]) - max(a[1], b[1])
    i = max(w, 0) * max(h, 0)
    return i / (ar(a) + ar(b) - i)
round(iou((0, 0, 10, 10), (5, 5, 15, 15)), 3)
# -> 0.143
from ultralytics import YOLO
# object detection [chk]
m = YOLO('yolo26n.pt')
# weights are downloaded [ref]
m.train(data='data.yaml', epochs=50)
# train (GPU advised) [chk]
m.predict('photo.jpg', conf=0.25) # detect [chk]
```

Optuna and NSGA-II

Ch. 57

```
import optuna # best x (true = 2)
optuna.logging.set_verbosity(40)
def f(t):
    return (t.suggest_float('x', -5, 5) - 2) ** 2
s = optuna.samplers.TPESampler(seed=0)
st = optuna.create_study(sampler=s)
st.optimize(f, 30)
round(st.best_params['x'], 1) # -> 2.0
# 2 goals = Pareto front
from pymoo.algorithms.moo.nsga2 import NSGA2
from pymoo.problems import get_problem
from pymoo.optimize import minimize
p = get_problem('zdt1')
r = minimize(p, NSGA2(pop_size=20), ('n_gen', 20), seed=1)
r.F.shape[1] # -> 2
```

SHAP and MLflow

Ch. 58

```
# synthetic data
from sklearn.datasets import make_classification
X, y = make_classification(200, random_state=0)
# each feature's share
import numpy as np, shap
from sklearn.ensemble import RandomForestClassifier
m = RandomForestClassifier(20, random_state=0)
e = shap.TreeExplainer(m.fit(X, y))
np.shape(e.shap_values(X[5])) # -> (5, 20, 2)
import mlflow # track experiments
mlflow.set_tracking_uri('sqlite:///m.db')
with mlflow.start_run():
    mlflow.log_metric('accuracy', 0.9)
r = mlflow.search_runs(search_all_experiments=True)
len(r) # -> 1
```

RAG, vectors, agents, MCP

Ch. 60, 62

```
# nearest neighbour
import faiss, numpy as np
v = np.random.default_rng(0).random((9, 4))
v = v.astype('float32')
ix = faiss.IndexFlatL2(4)
ix.add(v)
int(ix.search(v[:1], 2)[1][0][0]) # -> 0
import chromadb # vector database
c = chromadb.Client().create_collection('pond')
c.add(ids=['1', '2'], documents=['rice', 'fish'],
      embeddings=[[1, 0], [0, 1]])
q = dict(query_embeddings=[[1, .1]], n_results=1)
c.query(**q)['documents'] # -> [['rice']]
# RAG: context goes in the prompt
context, question = 'Pond A: pH 7.2', 'pH pond A?'
f'Answer from context only.\n{context}\n{question}'
# -> 'Answer from context only.\nPond A: pH 7.2\npH po...'
# resp = client.generate(prompt)
# needs a key [disamarkan]; not run [ref]
# agent: loop until done/limit
def agent(model, tool, limit=5):
    for _ in range(limit):
        a = model()
        if 'answer' in a:
            return a['answer']
        tool[a['tool']](a['arg'])
# a scripted model answers
s = iter([{'tool': 't', 'arg': 2}, {'answer': '5'}])
agent(lambda: next(s), {'t': lambda a: a + 3})
# -> '5'
# MCP server + client (in memory)
import asyncio
from mcp import Client
from mcp.server import MCPServer
sv = MCPServer('pond')
@sv.tool()
def biomass(n: int, g: float) -> float:
    return round(n * g / 1000, 2)
async def main():
    async with Client(sv) as k:
        a = {'n': 800, 'g': 540.5}
        h = await k.call_tool('biomass', a)
        return h.structured_content
asyncio.run(main()) # -> {'result': 432.4}
```

Descriptors, bytecode, internals

Ch. 64-65

```
# descriptor: attribute guard
class Positive:
    def __set_name__(self, o, n):
        self.n = '.' + n
    def __get__(self, o, t=None):
        return getattr(o, self.n)
    def __set__(self, o, v):
        if v <= 0:
            raise ValueError('must be > 0')
        setattr(o, self.n, v)
class Tree:
    d = Positive()
    def __init__(self, d):
        self.d = d
Tree(32.5).d # -> 32.5
type('B', (), {'x': 1}).x
# class made at run time -> 1
import dis # bytecode
f = lambda a: a + 1
[i.opname for i in dis.get_instructions(f)][-2:]
# -> ['BINARY_OP', 'RETURN_VALUE']
import sys # ref count (x + the argument)
x = []
sys.getrefcount(x) # -> 2
```

numba, Cython, CuPy

Ch. 66

```
# JIT: C-speed for-loops
from numba import njit
@njit
def sum_of_squares(n):
    s = 0
    for i in range(n):
        s += i * i
    return s
sum_of_squares(1000) # -> 332833500
$ cythonize -i ndvi_cy.pyx
# compile .pyx (needs a C compiler) [chk]
import cupy as cp # NVIDIA GPU like numpy
float(cp.asarray([1., 2., 3.]).sum()) # -> 6.0
```

Celery (task queue)

Ch. 67

```
# eager mode: test without a broker
from celery import Celery
app = Celery('t', broker='memory://',
            backend='cache+memory://')
app.conf.task_always_eager = True
@app.task
def add(a, b):
    return a + b
add.delay(2, 3).get() # -> 5
# automatic retries
@app.task(bind=True, max_retries=3)
def fragile(self, x):
    return x
fragile.delay(7).get() # -> 7
$ celery -A tugas_worker --concurrency=3
# three worker processes [chk]
redis.Redis.from_url(url, protocol=2)
# Redis 5 (needs a server) [ref]
```