

Karandeep Singh

Python basics, the DevOps cut.

Lists, tuples, loops, functions, classes, lambda, sorting. Many examples, one card.

LISTS: THE FLEET

```
servers = ["api-1", "api-2", "db-1"]
servers.append("api-3")   # add a node
servers.insert(0, "lb-1") # add at front
servers.remove("db-1")    # drop one
servers[0]  servers[-1]   # first, last
servers[:2]  # first two: a deploy batch
servers[::2] # every second server
len(servers) "api-1" in servers
```

TUPLES: FIXED RECORDS

```
inst = ("i-0abc", "ca-central-1", "running")
iid, region, state = inst  # unpack
inst[1]          # read by position
# inst[2] = "stopped"  # ERROR: fixed
def lo_hi(ages): return min(ages), max(ages)
lo, hi = lo_hi([12, 340, 7])  # returns tuple
point = 10, 20   # brackets optional
```

LOOPS: THE ROUNDS

```
for s in servers: print("check", s)
for i, s in enumerate(servers, 1):
    print(i, s)          # rollout order
for s in servers:
    if s == "db-1": break  # stop early
    if s == "lb-1": continue  # skip it
n = 0
while n < 5: n += 1      # retry limit
for s in servers: pass   # placeholder
```

FUNCTIONS

```
def is_healthy(code):
    return code == 200
def deploy(env, replicas=1):  # default
    return env, replicas
deploy("sandbox")        # uses default
deploy("live", replicas=3)
deploy(env="live", replicas=3)  # named
def total(*nums): return sum(nums)
total(1, 2, 3)   # any count of args
```

LAMBDA

```
double = lambda n: n * 2
double(21)
alive = lambda s: s["state"] == "running"
key=lambda p: p[1]   # sort by 2nd item
key=lambda s: s.lower()  # ignore case
key=lambda x: -x     # biggest first
# one line, one job: a key, not logic
```

SORTING

```
sorted(pods)             # new list
pods.sort()              # in place
sorted(pods, reverse=True)
sorted(pods, key=lambda p: p[1])
sorted(pods, key=lambda p: p[1],
       reverse=True)     # worst first
sorted(names, key=len)   # shortest first
min(pods)  max(pods)     # extremes free
```

CLASSES

```
class Server:
    def __init__(self, name, status):
        self.name = name
        self.status = status
    def is_live(self):
        return self.status == "running"
    def __str__(self):
        return self.name
api = Server("api-1", "running")
api.is_live()  print(api)  # api-1
```

COMPREHENSIONS + MORE

```
live = [s.name for s in fleet if s.is_live()]
ages = {s.name: s.age for s in fleet}
uniq = {s.region for s in fleet}  # set
any(s.is_live() for s in fleet)   # one?
all(s.is_live() for s in fleet)   # every?
sum(s.restarts for s in fleet)    # total
"web" if env == "live" else "api"
msg = f"{api.name} is {api.status}"
```

Basics are not abstract. They run on servers.

https://karandeepsingh.ca



Karandeep Singh

Python imports, the DevOps cut.

requests, json, yaml, click. Many examples, one card.

REQUESTS: GET

```
r = requests.get(url, timeout=10)
r = requests.get(url, timeout=10,
    params={"region": "ca-central-1"})
r.status_code        # 200
r.raise_for_status() # error if 4xx/5xx
data = r.json()      # parsed body
r.headers["X-Request-Id"]
```

REQUESTS: POST + SESSION

```
requests.post(url, timeout=10,
    json={"env": "live", "replicas": 3})
requests.post(url, data="raw", timeout=10)
requests.delete(url, timeout=10)
s = requests.Session()   # reuse it
s.headers.update({"Authorization": tok})
s.get(url)  s.post(url)  # headers kept
```

JSON: STRINGS

```
data = json.loads(text)   # text to dict
text = json.dumps(data)   # dict to text
json.dumps(data, indent=2) # readable
json.dumps(data, sort_keys=True)
data["Replicas"]          # must exist
data.get("Region", "ca-central-1")
```

JSON: FILES

```
cfg = json.load(open("cfg.json"))
json.dump(cfg, open("o.json", "w"),
          indent=2)
# loads/dumps: strings (s = string)
# load/dump: files
out = subprocess.run([...],
    capture_output=True, text=True)
json.loads(out.stdout)  # CLI output
```

YAML

```
import yaml  # pip install pyyaml
cfg = yaml.safe_load(open("values.yaml"))
cfg["deployment"]["replicas"]
yaml.safe_dump(cfg, open("o.yaml", "w"))
# the file itself:
deployment:
  image: myapi:1.4.2
  replicas: 3
# never yaml.load: config must not run code
```

CLICK

```
@click.command()
@click.option("--env", default="sandbox")
@click.option("--replicas", type=int,
              default=1)
@click.option("--dry-run", is_flag=True)
@click.argument("target")
def deploy(env, replicas, dry_run, target):
    click.echo(f"{target}: {env}")
# deploy.py web --env live --replicas 3
# deploy.py --help  # free help page
```

Config in, call out, answer back.

https://karandeepsingh.ca



Karandeep Singh

Python collections, the DevOps cut.

Counter, defaultdict, deque, namedtuple and friends. Many examples, one card.

COUNTER

```
c = Counter(["Running", "Running",
             "Pending", "Failed"])
c["Running"]          # 2
c.most_common(2)      # ranked pairs
c.total()             # sum of counts
c.update(["Failed"])  # add more
c["Missing"]          # 0, no KeyError
list(c.elements())    # counts as items
```

NAMEDTUPLE

```
Instance = namedtuple("Instance",
           ["id", "region", "state"])
db = Instance("i-0abc", "west", "running")
db.state   db[2]      # name or index
db._replace(state="stopped")  # new copy
db._asdict()          # to a dict
Point = namedtuple("P", "x y",
                   defaults=[0, 0])
```

DEFAULTDICT

```
by_region = defaultdict(list)
by_region["west"].append("api-1")
count = defaultdict(int)
count["errors"] += 1     # starts at 0
seen = defaultdict(set)
seen["west"].add("api-1")  # no dupes
plain = dict(by_region)   # convert back
```

MORE COLLECTIONS

```
ChainMap(prod, defaults)  # layered cfg
OrderedDict: move_to_end("key")
od.popitem(last=False)    # FIFO order
UserDict / UserList: custom dict/list
Counter(a) + Counter(b)   # add tallies
Counter(a) - Counter(b)   # day delta
Counter(a) & Counter(b)   # the overlap
```

DEQUE

```
jobs = deque(["backup", "snapshot"])
jobs.append("deploy")     # right end
jobs.appendleft("ping")   # left end
jobs.popleft()  jobs.pop()
jobs.rotate(1)   # last moves to front
tail = deque(maxlen=100)  # log tail
jobs[0]  len(jobs)  "deploy" in jobs
```

PUT TOGETHER

```
levels = Counter()
for line in open("app.log"):
    levels[line.split()[0]] += 1
print(levels.most_common())
by_host = defaultdict(Counter)
for line in open("app.log"):
    p = line.split()
    by_host[p[1]][p[0]] += 1  # per host
```

Counting by hand is a smell.

https://karandeepsingh.ca