Perfect! Here’s a **complete interview-ready sheet** combining **all 90 questions**:

It’s divided into **Coding Questions** (with ready-to-run Python code) and **Theory Questions** (concise, interview-style answers). You can use it as a **single-page reference sheet**.

**Python Interview Ready Sheet (1–90)**

**✅ Coding Questions (Ready-to-run Python)**

**Data Types, Lists, Strings, Loops, Functions, OOP, Iterators/Generators, Exception Handling**

# 1. Palindrome without slicing

def is\_palindrome(s):

n = len(s)

for i in range(n//2):

if s[i] != s[n-1-i]:

return False

return True

# 2. Remove duplicates without set

def remove\_duplicates(lst):

res = []

for i in lst:

if i not in res: res.append(i)

return res

# 3. Merge dictionaries

def merge\_dicts(d1, d2):

return {\*\*d1, \*\*d2}

# 4. Frequency of characters

from collections import Counter

def char\_freq(s): return dict(Counter(s))

# 5. Max without built-in

def max\_val(lst):

m = lst[0]

for x in lst: m = x if x > m else m

return m

# 6. Reverse list in-place

def reverse\_inplace(lst):

n = len(lst)

for i in range(n//2):

lst[i], lst[n-1-i] = lst[n-1-i], lst[i]

# 7. Flatten nested list

def flatten(lst):

res = []

for i in lst:

if isinstance(i, list): res.extend(i)

else: res.append(i)

return res

# 8. Count vowels & consonants

def count\_vowels\_consonants(s):

vowels = "aeiouAEIOU"

v, c = 0, 0

for ch in s:

if ch.isalpha():

if ch in vowels: v += 1

else: c += 1

return v, c

# 9. Second largest

def second\_largest(lst):

unique = list(set(lst))

unique.sort()

return unique[-2]

# 10. Rotate list by k

def rotate(lst, k):

k %= len(lst)

return lst[-k:] + lst[:-k]

# 11. Fibonacci n terms

def fib(n):

a, b = 0, 1

res = []

for \_ in range(n):

res.append(a)

a, b = b, a + b

return res

# 12. Multiplication table

def multiplication\_table(n):

for i in range(1, 11):

print(n, "x", i, "=", n\*i)

# 13. Primes up to n

def primes(n):

res = []

for i in range(2, n+1):

for j in range(2,int(i\*\*0.5)+1):

if i%j==0: break

else: res.append(i)

return res

# 14. Check anagrams

def is\_anagram(s1, s2):

return sorted(s1) == sorted(s2)

# 15. Numbers divisible by 3 & 5

def div\_3\_5(): return [i for i in range(1,101) if i%3==0 and i%5==0]

# 16. \*args and \*\*kwargs

def args\_kwargs(\*args, \*\*kwargs):

print("args:", args)

print("kwargs:", kwargs)

# 17. Recursive factorial

def factorial(n): return 1 if n==0 else n\*factorial(n-1)

# 18. Recursive Fibonacci

def fib\_rec(n):

if n==0: return 0

if n==1: return 1

return fib\_rec(n-1)+fib\_rec(n-2)

# 19. Unique elements

def unique\_list(lst):

res=[]

for i in lst:

if i not in res: res.append(i)

return res

# 20. Sum of digits

def sum\_digits(n):

s = 0

while n>0: s+=n%10; n//=10

return s

# 21. BankAccount class

class BankAccount:

def \_\_init\_\_(self, balance=0): self.balance=balance

def deposit(self,a): self.balance+=a

def withdraw(self,a):

if a<=self.balance: self.balance-=a

else: print("Insufficient funds")

def check\_balance(self): return self.balance

# 28. Fibonacci generator

def fib\_gen(n):

a,b=0,1

for \_ in range(n): yield a; a,b=b,a+b

# 29. File line generator

def read\_file\_lines(filename):

with open(filename) as f:

for line in f: yield line.strip()

# 30. Prime generator

def prime\_gen(n):

for num in range(2,n+1):

for i in range(2,int(num\*\*0.5)+1):

if num%i==0: break

else: yield num

# 31. Division by zero handling

try: print(5/0)

except ZeroDivisionError: print("Cannot divide by zero")

# 33. Retry division

def safe\_div(a,b):

for \_ in range(3):

try: return a/b

except ZeroDivisionError: print("Retrying...")

return None

# 35. Safe file reader

try:

with open("data.txt") as f: print(f.read())

except FileNotFoundError: print("File not found")

**Advanced Coding (18,25,33,34,41–44,46–50,71,72,76,81,82,84,85)**

# 18. Async fetch multiple URLs

import asyncio, aiohttp

async def fetch(url):

async with aiohttp.ClientSession() as s:

async with s.get(url) as r: return await r.text()

async def fetch\_all(urls): await asyncio.gather(\*[fetch(u) for u in urls])

# 25. Manual LRU cache

from collections import OrderedDict

class LRUCache:

def \_\_init\_\_(self, capacity): self.capacity=capacity; self.cache=OrderedDict()

def get(self,key):

if key in self.cache: self.cache.move\_to\_end(key); return self.cache[key]

return -1

def put(self,key,val):

if key in self.cache: self.cache.move\_to\_end(key)

self.cache[key]=val

if len(self.cache)>self.capacity: self.cache.popitem(last=False)

# 33. Singleton

class Singleton:

\_instance=None

def \_\_new\_\_(cls,\*a,\*\*kw):

if not cls.\_instance: cls.\_instance=super().\_\_new\_\_(cls)

return cls.\_instance

# 34. Factory pattern

class Shape:

def draw(self): pass

class Circle(Shape):

def draw(self): print("Circle")

class Square(Shape):

def draw(self): print("Square")

class ShapeFactory:

@staticmethod

def create\_shape(name):

if name=="circle": return Circle()

elif name=="square": return Square()

return None

# 41. Trie

class TrieNode:

def \_\_init\_\_(self): self.children={}; self.is\_end=False

class Trie:

def \_\_init\_\_(self): self.root=TrieNode()

def insert(self,word):

node=self.root

for ch in word:

if ch not in node.children: node.children[ch]=TrieNode()

node=node.children[ch]

node.is\_end=True

def search(self,word):

node=self.root

for ch in word:

if ch not in node.children: return False

node=node.children[ch]

return node.is\_end

# 42. Thread-safe queue

import queue

q=queue.Queue(); q.put(1); item=q.get()

# 43. Linked list

class Node:

def \_\_init\_\_(self,data): self.data=data; self.next=None

class LinkedList:

def \_\_init\_\_(self): self.head=None

def insert(self,data):

n=Node(data); n.next=self.head; self.head=n

def delete(self,key):

prev=None; curr=self.head

while curr:

if curr.data==key:

if prev: prev.next=curr.next

else: self.head=curr.next

return

prev,curr=curr,curr.next

def search(self,key):

curr=self.head

while curr:

if curr.data==key: return True

curr=curr.next

return False

# 44. Priority queue

import heapq

pq=[]; heapq.heappush(pq,(2,"task2")); heapq.heappush(pq,(1,"task1"))

priority, task=heapq.heappop(pq)

# 46–47. Graph BFS/DFS

from collections import deque

def bfs(graph,start):

visited=set(); q=deque([start])

while q:

n=q.popleft()

if n not in visited:

print(n); visited.add(n)

q.extend(graph.get(n,[]))

def dfs(graph,start,visited=None):

if visited is None: visited=set()

visited.add(start); print(start)

for nb in graph.get(start,[]):

if nb not in visited: dfs(graph,nb,visited)

# 49. Rate limiter

import time

from collections import deque

class RateLimiter:

def \_\_init\_\_(self,max\_calls,period): self.calls=deque(); self.max\_calls=max\_calls; self.period=period

def allow(self):

now=time.time()

while self.calls and self.calls[0]<=now-self.period: self.calls.popleft()

if len(self.calls)<self.max\_calls: self.calls.append(now); return True

return False

# 50. Distributed counter

from multiprocessing import Value, Lock

class DistributedCounter:

def \_\_init\_\_(self): self.value=Value('i',0); self.lock=Lock()

def increment(self):

with self.lock: self.value.value+=1; return self.value.value

# 71. REST API without Flask

from http.server import BaseHTTPRequestHandler,HTTPServer

import json

class SimpleHandler(BaseHTTPRequestHandler):

def do\_GET(self):

self.send\_response(200); self.send\_header("Content-Type","application/json"); self.end\_headers()

self.wfile.write(json.dumps({"message":"Hello"}).encode())

# 72. WebSockets (example)

import asyncio, websockets

async def echo(ws, path):

async for msg in ws: await ws.send(f"Echo: {msg}")

# 76. Celery task (pseudo)

# from celery import Celery

# app = Celery('tasks', broker='redis://localhost:6379/0')

# @app.task

# def add(x,y): return x+y

# 81. Custom context manager

class MyContext:

def \_\_init\_\_(self,f,m): self.file=open(f,m)

def \_\_enter\_\_(self): return self.file

def \_\_exit\_\_(self,exc\_type,exc\_val,exc\_tb): self.file.close()

# 82. Decorator class with params

class Decorator:

def \_\_init\_\_(self,prefix): self.prefix=prefix

def \_\_call\_\_(self,func):

def wrapper(\*a,\*\*kw): print(self.prefix,"before call"); return func(\*a,\*\*kw)

return wrapper

# 84. Operator overloading

class Vector:

def \_\_init\_\_(self,x,y): self.x,self.y=x,y

def \_\_add\_\_(self,other): return Vector(self.x+other.x,self.y+other.y)

def \_\_repr\_\_(self): return f"Vector({self.x},{self.y})"

# 85. Lazy evaluation

def lazy\_range(n):

i=0

while i<n: yield i; i+=1

**✅ Theory Questions (Concise Answers)**

**Advanced Internals & Performance:**

1. Python uses reference counting + garbage collection in private heap.
2. id() → memory address, hash() → dict/set key, is → object identity.
3. GC cleans objects with zero references; handles cycles via generational GC.
4. Reference cycles are objects referring each other; GC detects and removes.
5. deepcopy → recursive copy; pickle → serialize/deserialize.
6. Assignment → reference, shallow → new object same nested objects, deep → full copy.
7. Lists → dynamic arrays, contiguous memory.
8. Dict → hash tables, open addressing.
9. OrderedDict → methods like move\_to\_end; dict 3.7+ preserves order.
10. Profiling → cProfile, timeit, line\_profiler.

**Concurrency & Async:**  
11. GIL → one thread executes Python bytecode; ensures thread safety but limits CPU-bound tasks.  
12. Thread → shared memory, light; Process → independent, true parallelism.  
13. Threading → I/O-bound; Multiprocessing → CPU-bound.  
14. asyncio → single-threaded async; threads → preemptive.  
15. Coroutines → async, Generators → lazy sequences.  
16. ThreadPool → threads; ProcessPool → processes.  
17. Cooperative → tasks yield; preemptive → OS interrupts.  
19. Race → concurrent access; Deadlock → circular wait; solve via locks.  
20. IPC → pipes, queues, shared memory, sockets.