Python Data & Algorithms Reference

NumPy • Pandas • Common Algorithms

NumPy Arrays - Basics

Operation	Code Example	Notes
Import	`import numpy as np`	Standard convention
Create array	`np.array([1, 2, 3])` `np.array([[1, 2], [3, 4]])`	1D and 2D arrays

Zeros/Ones	`np.zeros((3, 4))`
	`np.ones((2, 3))`
	`np.full((2, 2), 7)`
		Create filled arrays

Range	`np.arange(0, 10, 2)  # 0,2,4,6,8` `np.linspace(0, 1, 5)  # 5 points`	arange: step, linspace: count

Random	`np.random.rand(3, 2)  # uniform [0,1)`
	`np.random.randn(3, 2)  # normal`
	`np.random.randint(0, 10, (3, 2))`
		Random arrays

Identity	`np.eye(3)` `np.identity(3)`	3x3 identity matrix

Shape/Size	`arr.shape  # (3, 4)`
	`arr.size  # 12`
	`arr.ndim  # 2`
		Dimensions info

Dtype	`arr.dtype` `arr.astype(np.float32)`	Data type info/convert

Reshape	`arr.reshape(2, 6)`
	`arr.flatten()`
	`arr.ravel()`
		Change dimensions

NumPy - Indexing & Operations

Operation	Code Example	Notes

Indexing	`arr[0]  # First element`
	`arr[1, 2]  # Row 1, Col 2`
	`arr[-1]  # Last`
		Zero-indexed

Slicing	`arr[1:4]  # Elements 1-3`
	`arr[:, 0]  # All rows, col 0`
	`arr[::2]  # Every 2nd`
		: means all, ::step

Boolean mask	`arr[arr > 5]` `arr[(arr > 5) & (arr < 10)]`	Filter by condition
Where	`np.where(arr > 5, arr, 0)` `np.where(condition)[0]`	Conditional replacement

Arithmetic	`arr + 5`
	`arr * 2`
	`arr1 + arr2`
		Element-wise ops

Matrix ops	`arr1 @ arr2  # Matrix multiply`
	`arr1.dot(arr2)`
	`np.matmul(arr1, arr2)`
		Linear algebra

Aggregation	`arr.sum()`
	`arr.mean()`
	`arr.std()`
	`arr.min()`
	`arr.max()`
		Statistics

Axis ops	`arr.sum(axis=0)  # Column sum` `arr.mean(axis=1)  # Row mean`	axis=0: cols, axis=1: rows

Sorting	`np.sort(arr)`
	`arr.argsort()  # indices`
	`np.partition(arr, 3)`
		Sort operations

Stacking	`np.vstack([arr1, arr2])`
	`np.hstack([arr1, arr2])`
	`np.concatenate([arr1, arr2])`
		Combine arrays

Pandas DataFrames - Creation & Basics

Operation	Code Example	Notes
Import	`import pandas as pd`	Standard convention
From dict	`pd.DataFrame({"A": [1,2], "B": [3,4]})`	Dict of lists/arrays
From array	`pd.DataFrame(arr, columns=["A", "B"])`	NumPy array to DF
From CSV	`pd.read_csv("file.csv")` `pd.read_csv("file.csv", sep="\t")`	Read CSV/TSV
From Excel	`pd.read_excel("file.xlsx", sheet_name="Sheet1")`	Read Excel
From JSON	`pd.read_json("file.json")`	Read JSON
To CSV	`df.to_csv("out.csv", index=False)`	Write CSV
To Excel	`df.to_excel("out.xlsx", index=False)`	Write Excel

Info	`df.info()`
	`df.describe()`
	`df.dtypes`
		DataFrame summary

Shape	`df.shape  # (rows, cols)` `len(df)  # rows`	Dimensions
Head/Tail	`df.head(10)` `df.tail(10)`	First/last n rows
Columns	`df.columns` `df.columns = ["A", "B", "C"]`	Get/set column names

Pandas - Selection & Filtering

Operation	Code Example	Notes

Select column	`df["A"]`
	`df.A`
	`df[["A", "B"]]`
		Single or multiple cols

Select rows	`df[0:5]  # Slice`
	`df.iloc[0:5]  # Position`
	`df.loc[0:5]  # Label`
		iloc: position, loc: label

iloc	`df.iloc[0]  # First row`
	`df.iloc[0, 1]  # Row 0, Col 1`
	`df.iloc[:, 0:2]`
		Integer position-based

loc	`df.loc[0]  # Label index 0` `df.loc[:, "A":"C"]`	Label-based indexing
Boolean filter	`df[df["A"] > 5]` `df[(df["A"] > 5) & (df["B"] < 10)]`	Filter rows by condition
isin	`df[df["A"].isin([1, 2, 3])]`	Filter by value list
query	`df.query("A > 5 and B < 10")`	SQL-like filtering
sample	`df.sample(n=10)` `df.sample(frac=0.1)`	Random sample

unique	df["A"].unique() df["A"].nunique()	Unique values/count
value_counts	df["A"].value_counts()	Count occurrences

Pandas - Data Manipulation

Operation	Code Example	Notes
Add column	df["C"] = df["A"] + df["B"] df["D"] = 10	New column
Drop column	df.drop("A", axis=1) df.drop(columns=["A", "B"])	axis=1 for columns
Drop row	df.drop([0, 1]) df.drop(index=[0, 1])	By index
Rename	df.rename(columns={"A": "X"})	Rename columns
Sort	df.sort_values("A") df.sort_values(["A", "B"], ascending=False)	Sort by values
Replace	df.replace(0, np.nan) df["A"].replace({1: 10, 2: 20})	Replace values
Fill NA	df.fillna(0) df.fillna(method="ffill") df.fillna(df.mean())	Handle missing data
Drop NA	df.dropna() df.dropna(axis=1) df.dropna(subset=["A"])	Remove NA rows/cols
Apply	df["A"].apply(lambda x: x*2) df.apply(np.sum, axis=0)	Apply function
Map	df["A"].map({1: "a", 2: "b"})	Map values
Astype	df["A"].astype(int) df.astype({"A": int, "B": float})	Convert types

Pandas - GroupBy & Aggregation

Operation	Code Example	Notes
GroupBy	`df.groupby("A").sum()` `df.groupby(["A", "B"]).mean()`	Group and aggregate
Agg multiple	`df.groupby("A").agg({"B": "sum", "C": "mean"})`	Different aggs per col
Agg list	`df.groupby("A")["B"].agg(["sum", "mean", "std"])`	Multiple aggs on col
Transform	`df.groupby("A")["B"].transform("mean")`	Keep original shape
Filter groups	`df.groupby("A").filter(lambda x: len(x) > 5)`	Filter by group size
Pivot	`df.pivot(index="A", columns="B", values="C")`	Reshape wide
Pivot table	`df.pivot_table(values="C", index="A", columns="B", aggfunc="mean")`	With aggregation
Melt	`df.melt(id_vars=["A"], value_vars=["B", "C"])`	Reshape long
Crosstab	`pd.crosstab(df["A"], df["B"])`	Cross-tabulation

Pandas - Merging & Joining

Operation	Code Example	Notes
Concat	`pd.concat([df1, df2])` `pd.concat([df1, df2], axis=1)`	Stack vertically/horizontally
Merge (inner)	`pd.merge(df1, df2, on="key")`	SQL-like inner join
Merge (left)	`pd.merge(df1, df2, on="key", how="left")`	Keep all left rows
Merge (outer)	`pd.merge(df1, df2, on="key", how="outer")`	Keep all rows
Merge multiple	`pd.merge(df1, df2, on=["key1", "key2"])`	Multiple key columns
Join	`df1.join(df2, on="key", how="left")`	Index-based join
Append	`df1.append(df2, ignore_index=True)`	Add rows (deprecated, use concat)

Pandas - Time Series

Operation	Code Example	Notes
To datetime	`pd.to_datetime(df["date"])` `pd.to_datetime("2024-01-01")`	Parse dates

Date parts	`df["date"].dt.year`
	`df["date"].dt.month`
	`df["date"].dt.day`
		Extract components

Resample	`df.resample("D").sum()  # Daily` `df.resample("M").mean()  # Monthly`	Aggregate by time
Rolling	`df["A"].rolling(window=7).mean()`	Moving average
Shift	`df["A"].shift(1)  # Lag 1` `df["A"].shift(-1)  # Lead 1`	Shift values

Diff	df["A"].diff() # First difference	Change from previous
Date range	pd.date_range("2024-01-01", periods=10, freq="D")	Generate date range

Algorithms - Sorting & Searching

Algorithm	Code Example	Complexity

Binary Search	`def binary_search(arr, target):`
	`  left, right = 0, len(arr) - 1`
	`  while left <= right:`
	`  mid = (left + right) // 2`
	`  if arr[mid] == target:`
	`  return mid`
	`  elif arr[mid] < target:`
	`  left = mid + 1`
	`  else:`
	`  right = mid - 1`
	`  return -1`
		O(log n)
		Requires sorted array

Quick Sort	`def quicksort(arr):`
	`  if len(arr) <= 1:`
	`  return arr`
	`  pivot = arr[len(arr) // 2]`
	`  left = [x for x in arr if x < pivot]`
	`  middle = [x for x in arr if x == pivot]`
	`  right = [x for x in arr if x > pivot]`
	`  return quicksort(left) + middle + quicksort(right)`
		O(n log n) avg
		O(n²) worst

Merge Sort	`def merge_sort(arr):`
	`  if len(arr) <= 1:`
	`  return arr`
	`  mid = len(arr) // 2`
	`  left = merge_sort(arr[:mid])`
	`  right = merge_sort(arr[mid:])`
	`  return merge(left, right)`
	
	`def merge(left, right):`
	`  result = []`
	`  i = j = 0`
	`  while i < len(left) and j < len(right):`
	`  if left[i] <= right[j]:`
	`  result.append(left[i])`
	`  i += 1`
	`  else:`
	`  result.append(right[j])`
	`  j += 1`
	`  result.extend(left[i:])`
	`  result.extend(right[j:])`
	`  return result`
		O(n log n)
		Stable sort

Heap Sort	`import heapq`
	
	`def heap_sort(arr):`
	`  heapq.heapify(arr)`
	`  return [heapq.heappop(arr) for _ in range(len(arr))]`
		O(n log n)
		In-place

Algorithms - Data Structures

Structure	Code Example	Use Case

Stack	`class Stack:`
	`  def __init__(self):`
	`  self.items = []`
	`  def push(self, item):`
	`  self.items.append(item)`
	`  def pop(self):`
	`  return self.items.pop()`
	`  def is_empty(self):`
	`  return len(self.items) == 0`
	
	`# Or just use list:`
	`stack = []`
	`stack.append(1)  # push`
	`stack.pop()  # pop`
		LIFO, DFS, undo/redo, balanced parens

Queue	`from collections import deque`
	
	`queue = deque()`
	`queue.append(1)  # enqueue`
	`queue.popleft()  # dequeue`
	
	`# Or use queue.Queue for thread-safe:`
	`from queue import Queue`
	`q = Queue()`
	`q.put(1)`
	`q.get()`
		FIFO, BFS, task scheduling

Heap	`import heapq`
	
	`heap = []`
	`heapq.heappush(heap, 3)`
	`heapq.heappush(heap, 1)`
	`heapq.heappush(heap, 2)`
	`min_val = heapq.heappop(heap)  # 1`
	
	`# Max heap (negate values):`
	`heapq.heappush(heap, -x)`
	`max_val = -heapq.heappop(heap)`
		Priority queue, top K elements, Dijkstra

Hash Table	`# Built-in dict`
	`hash_map = {}`
	`hash_map["key"] = "value"`
	`if "key" in hash_map:`
	`  value = hash_map["key"]`
	
	`# Counter for frequencies:`
	`from collections import Counter`
	`freq = Counter([1, 2, 2, 3, 3, 3])`
	`# Counter({3: 3, 2: 2, 1: 1})`
		O(1) lookup, counting, deduplication

Linked List	class Node:
	    def __init__(self, data):
	        self.data = data
	        self.next = None
	
	class LinkedList:
	    def __init__(self):
	        self.head = None
	
	    def append(self, data):
	        if not self.head:
	            self.head = Node(data)
	        else:
	            curr = self.head
	            while curr.next:
	                curr = curr.next
	            curr.next = Node(data)
		Dynamic size,
		fast insert/delete,
		LRU cache

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 char in word:
	            if char not in node.children:
	                node.children[char] = TrieNode()
	            node = node.children[char]
	        node.is_end = True
		Prefix search,
		autocomplete,
		spell check

Algorithms - Graph Traversal

Algorithm	Code Example	Complexity

BFS	`from collections import deque`
	
	`def bfs(graph, start):`
	`  visited = set()`
	`  queue = deque([start])`
	`  visited.add(start)`
	
	`  while queue:`
	`  vertex = queue.popleft()`
	`  print(vertex)`
	
	`  for neighbor in graph[vertex]:`
	`  if neighbor not in visited:`
	`  visited.add(neighbor)`
	`  queue.append(neighbor)`
		O(V + E)
		Level-by-level,
		shortest path

DFS	`def dfs(graph, start, visited=None):`
	`  if visited is None:`
	`  visited = set()`
	`  visited.add(start)`
	`  print(start)`
	
	`  for neighbor in graph[start]:`
	`  if neighbor not in visited:`
	`  dfs(graph, neighbor, visited)`
	`  return visited`
	
	`# Iterative:`
	`def dfs_iterative(graph, start):`
	`  visited = set()`
	`  stack = [start]`
	
	`  while stack:`
	`  vertex = stack.pop()`
	`  if vertex not in visited:`
	`  visited.add(vertex)`
	`  print(vertex)`
	`  stack.extend(graph[vertex])`
		O(V + E)
		Goes deep,
		backtracking

Dijkstra	import heapq
	
	def dijkstra(graph, start):
	  distances = {node: float("inf") for node in graph}
	  distances[start] = 0
	  pq = [(0, start)]
	
	  while pq:
	  curr_dist, curr_node = heapq.heappop(pq)
	
	  if curr_dist > distances[curr_node]:
	  continue
	
	  for neighbor, weight in graph[curr_node].items():
	  distance = curr_dist + weight
	  if distance < distances[neighbor]:
	  distances[neighbor] = distance
	  heapq.heappush(pq, (distance, neighbor))
	
	  return distances
		O((V+E) log V)
		Shortest path,
		weighted graph

Topological Sort	def topological_sort(graph):
	  def dfs(node):
	  visited.add(node)
	  for neighbor in graph.get(node, []):
	  if neighbor not in visited:
	  dfs(neighbor)
	  stack.append(node)
	
	  visited = set()
	  stack = []
	
	  for node in graph:
	  if node not in visited:
	  dfs(node)
	
	  return stack[::-1]
		O(V + E)
		DAG ordering,
		dependencies

Algorithms - Dynamic Programming

Problem	Code Example	Pattern

Fibonacci	def fib(n, memo={}):
	    if n in memo:
	        return memo[n]
	    if n <= 1:
	        return n
	    memo[n] = fib(n-1, memo) + fib(n-2, memo)
	    return memo[n]
	
	# Bottom-up:
	def fib_dp(n):
	    if n <= 1:
	        return n
	    dp = [0] * (n + 1)
	    dp[1] = 1
	    for i in range(2, n + 1):
	        dp[i] = dp[i-1] + dp[i-2]
	    return dp[n]
		Memoization
		Bottom-up

Longest Common Subsequence	def lcs(s1, s2):
	    m, n = len(s1), len(s2)
	    dp = [[0] * (n + 1) for _ in range(m + 1)]
	
	    for i in range(1, m + 1):
	        for j in range(1, n + 1):
	            if s1[i-1] == s2[j-1]:
	                dp[i][j] = dp[i-1][j-1] + 1
	            else:
	                dp[i][j] = max(dp[i-1][j], dp[i][j-1])
	
	    return dp[m][n]
		2D DP table
		String matching

0/1 Knapsack	def knapsack(weights, values, capacity):
	    n = len(weights)
	    dp = [[0] * (capacity + 1) for _ in range(n + 1)]
	
	    for i in range(1, n + 1):
	        for w in range(capacity + 1):
	            if weights[i-1] <= w:
	                dp[i][w] = max(
	                    values[i-1] + dp[i-1][w - weights[i-1]],
	                    dp[i-1][w]
	                )
	            else:
	                dp[i][w] = dp[i-1][w]
	
	    return dp[n][capacity]
		2D DP
		Optimization

Coin Change	def coin_change(coins, amount):
	    dp = [float("inf")] * (amount + 1)
	    dp[0] = 0
	
	    for coin in coins:
	        for x in range(coin, amount + 1):
	            dp[x] = min(dp[x], dp[x - coin] + 1)
	
	    return dp[amount] if dp[amount] != float("inf") else -1
		1D DP
		Min coins

Common Algorithm Patterns

Pattern	Code Example	When to Use

Two Pointers	def two_sum_sorted(arr, target):
	    left, right = 0, len(arr) - 1
	    while left < right:
	        current_sum = arr[left] + arr[right]
	        if current_sum == target:
	            return [left, right]
	        elif current_sum < target:
	            left += 1
	        else:
	            right -= 1
	    return []
		Sorted arrays,
		pairs, palindromes

Sliding Window	def max_sum_subarray(arr, k):
	    window_sum = sum(arr[:k])
	    max_sum = window_sum
	
	    for i in range(k, len(arr)):
	        window_sum = window_sum - arr[i-k] + arr[i]
	        max_sum = max(max_sum, window_sum)
	
	    return max_sum
		Subarrays,
		substrings,
		contiguous

Fast & Slow Ptr	def has_cycle(head):
	    slow = fast = head
	    while fast and fast.next:
	        slow = slow.next
	        fast = fast.next.next
	        if slow == fast:
	            return True
	    return False
		Cycle detection,
		middle of list

Backtracking	def permute(nums):
	    result = []
	
	    def backtrack(path):
	        if len(path) == len(nums):
	            result.append(path[:])
	            return
	
	        for num in nums:
	            if num in path:
	                continue
	            path.append(num)
	            backtrack(path)
	            path.pop()
	
	    backtrack([])
	    return result
		Permutations,
		combinations,
		N-queens

Big-O Complexity Cheatsheet

Notation	Name	Example Operations
O(1)	Constant	Array access, hash table lookup, dict get/set
O(log n)	Logarithmic	Binary search, balanced tree operations
O(n)	Linear	Array traversal, single loop, linear search
O(n log n)	Linearithmic	Merge sort, quick sort (average), heap sort
O(n²)	Quadratic	Nested loops, bubble sort, selection sort
O(n³)	Cubic	Triple nested loops, matrix multiplication
O(2■)	Exponential	Recursive fibonacci, power set generation
O(n!)	Factorial	Permutations, traveling salesman (brute force)

Data Structure Time Complexities

Data Structure	Access	Search	Insert	Delete	Space
Array	O(1)	O(n)	O(n)	O(n)	O(n)
Linked List	O(n)	O(n)	O(1)	O(1)	O(n)
Stack	O(n)	O(n)	O(1)	O(1)	O(n)
Queue	O(n)	O(n)	O(1)	O(1)	O(n)
Hash Table	N/A	O(1)*	O(1)*	O(1)*	O(n)
Binary Tree	O(n)	O(n)	O(n)	O(n)	O(n)
BST (balanced)	O(log n)	O(log n)	O(log n)	O(log n)	O(n)
Heap	O(n)	O(n)	O(log n)	O(log n)	O(n)
Trie	O(k)	O(k)	O(k)	O(k)	O(n×k)

* Average case. Worst case is O(n). k = key length