1) Title: Console I/O Formatter (Advanced)

Overview: Design robust console utilities for reading, parsing, and formatting tabular text for display and logs.

Functionalities to be designed

Functionality	Input	Output	Description	Constraints	
`read_ints_from_stdin`	single line from stdin	`list[int]`	Split by spaces, cast to `int`	Invalid tokens raise `ValueError`	
`read_floats_from_stdin`	single line from stdin	`list[float]`	Split by spaces, cast to `float`	Invalid tokens raise `ValueError`	
`format_table_row`	`list[str]`, `int` width	`str`	Left-align each cell to width, join with `"	"`	Width ≥ 1; pad shorter cells
`format_table_auto_widths`	`list[list[str]]` rows	`list[str]`	Compute per-column widths, return formatted rows	At least one row; columns uniform length	
`right_align_numbers`	`list[str]`, width	`str`	Right-align numeric substrings, left-align text	Numeric detection via `str.isdigit()`	
`safe_echo_count`	`list[str]`	`str`	Return `"Count = <n>"` for items	Works with any list	

Test cases

1. Input: `read_ints_from_stdin` with `"10 20 -3"` → Output: `[10, 20, -3]` (split and cast)
2. Input: `read_floats_from_stdin` with `"1.5 2.0"` → Output: `[1.5, 2.0]` (floats parsed)
3. Input: `format_table_row(["Alice","42"], 6)` → Output: `"Alice | 42 "` (padding)
4. Input: `safe_echo_count(["a","b"])` → Output: `"Count = 2"` (count)

---

2) Title: File I/O: Robust Read/Transform/Write

Overview: Implement file utilities with encoding handling, atomic writes, chunk reading, and line filtering.

Functionalities to be designed

Functionality	Input	Output	Description	Constraints
`read_lines`	path `str`, encoding `str`	`list[str]`	Read lines, strip whitespace, drop empty	File must exist; default UTF-8
`write_lines_atomic`	path `str`, `list[str]`, encoding `str`	`None`	Write via temp file then move	Overwrites destination atomically
`read_chunks`	path `str`, chunk_size `int`	`list[bytes]`	Read file in fixed-size chunks	chunk_size > 0
`tail_lines`	path `str`, n `int`	`list[str]`	Return last `n` stripped lines	n ≥ 0
`checksum_sha256`	path `str`	`str`	Compute SHA-256 hex digest	File must be readable
`filter_lines`	`list[str]`, predicate `Callable[[str],bool]`	`list[str]`	Keep lines matching predicate	Preserve order

Test cases

1. Input: `read_lines("input.txt", "utf-8")` with `"a\n\nb \n"` → `["a","b"]` (stripped, empty removed)
2. Input: `tail_lines("log.txt", 2)` from `["L1","L2","L3"]` → `["L2","L3"]` (last two)
3. Input: `filter_lines(["foo","bar"], lambda s: "a" in s)` → `["bar"]` (predicate)
4. Input: `write_lines_atomic("out.txt", ["X","Y"], "utf-8")` → `None` (atomically written)

---

3) Title: Type Coercion & Validation Suite

Overview: Design robust converters and validators across primitive types with failure handling.

Functionalities to be designed

Functionality	Input	Output	Description	Constraints			
to_int_safe	`str	float	int`	`int	None`	Convert to int; None if invalid	No exceptions; strip strings

Functionality	Input	Output	Description	Constraints			
`to_float_safe`	`str	int	float`	`float	None`	Convert to `float`; `None` if invalid	No exceptions
`normalize_bool`	`str	int	bool`	`bool	None`	`"true"/"false"`, `1/0`, `True/False`	Case-insensitive; else `None`
`parse_list_of_ints`	`str`	`list[int]	None`	Split by commas, convert to ints	Return `None` if any token invalid		
`is_numeric_str`	`str`	`bool`	Checks decimal integer string	No signs, no spaces			
`coerce_dict_values`	`dict[str,str]`, schema `dict[str,Callable]`	`dict[str,Any]	None`	Apply functions per key	Fail if any conversion fails		

Test cases

1. Input: `to_int_safe(" 42 ")` → `42` (trim then cast)
2. Input: `normalize_bool("False")` → `False` (case-insensitive)
3. Input: `parse_list_of_ints("1,2, x")` → `None` (invalid token)
4. Input: `coerce_dict_values({"a":"1","b":"3.5"}, {"a":int,"b":float})` → `{"a":1,"b":3.5}`

---

4) Title: Data Type Introspection & Immutability

Overview: Tools for checking hashability, iterability, and freezing structures.

Functionalities to be designed

Functionality	Input	Output	Description	Constraints
`is_hashable`	`Any`	`bool`	Return `True` if object is hashable	Use `collections.abc.Hashable`
`describe_type`	`Any`	`str`	Human-readable type name	`type(obj).__name__`
`is_iterable_non_str`	`Any`	`bool`	Iterable and not a `str`	Use `iter()` check
`freeze_list_shallow`	`list[Any]`	`tuple[Any,...]`	Shallow freeze a list	No deep copy
`deep_freeze`	`Any`	`Any`	Recursively freeze lists/sets/dicts to tuples	Dict becomes tuple of `(k,v)` pairs
`unfreeze_to_mutable`	`Any`	`Any`	Convert frozen structures back	Tuples → lists; pairs → dict

Test cases

1. Input: `is_hashable((1,2))` → `True` (tuple of hashables)
2. Input: `is_iterable_non_str("abc")` → `False` (string excluded)
3. Input: `freeze_list_shallow([1,2,3])` → `(1,2,3)` (shallow)
4. Input: `deep_freeze({"a":[1,2], "b":{3}})` → `(("a",(1,2)),("b",(3,)))`

---

5) Title: Control Flow: Grading with Rules

Overview: Compute grades with validations, curves, weights, and categorical outputs.

Functionalities to be designed

Functionality	Input	Output	Description	Constraints
`validate_scores`	`list[int]`	`bool`	All scores ∈ [0,100]	Empty allowed
`drop_lowest_k`	`list[int]`, `int` k	`list[int]`	Remove k smallest scores	k ≥ 0; if k ≥ len → `[]`
`weighted_average`	`list[int]`, `list[float]`	`float`	Weighted avg of scores	len(weights) == len(scores)
`grade_letter_pm`	`float` avg	`str`	`A+/A/A- ... F`	Exact thresholds defined
`pass_fail_boundary`	`float` avg, `float` cut	`bool`	`True` if avg ≥ cut	cut ∈ [0,100]
`histogram_buckets`	`list[int]`, bucket_size `int`	`dict[str,int]`	Count ranges like `0-9`, `10-19`	bucket_size divides 100

Test cases

1. Input: `drop_lowest_k([50, 80, 70], 1)` → `[80,70]` (removed 50)
2. Input: `weighted_average([80,90], [0.4,0.6])` → `86.0` (weighted)
3. Input: `grade_letter_pm(92.0)` → `"A-"` (thresholds)
4. Input: `histogram_buckets([5, 15, 27], 10)` → `{"0-9":1,"10-19":1,"20-29":1}`

---

6) Title: Control Flow: Retry + Circuit Breaker

Overview: Create retry logic with exponential backoff, jitter, limits, and circuit breaker states.

Functionalities to be designed

Functionality	Input	Output	Description	Constraints
`should_retry`	attempt `int`, max_attempts `int`	`bool`	Continue if attempt < max	attempt starts at 0
`next_delay_seconds`	attempt `int`, base `float`, cap `float`	`float`	`min(base * 2**attempt, cap)`	No sleep; compute only
`add_jitter`	delay `float`, jitter `float`	`float`	Adds random jitter ± amount	Deterministic via seed param
`max_total_time_exceeded`	elapsed `float`, limit `float`	`bool`	Stop if elapsed ≥ limit	Non-negative
`record_attempt`	`list[float]` delays, new_delay `float`	`list[float]`	Append delay for audit	Returns new list
`circuit_breaker_state`	failures `int`, threshold `int`	`str`	`"closed"`/`"open"` based on threshold	threshold ≥ 1

Test cases

1. Input: `should_retry(2,3)` → `True` (2 < 3)
2. Input: `next_delay_seconds(3, 0.5, 60.0)` → `4.0` (cap not reached)
3. Input: `max_total_time_exceeded(61.0, 60.0)` → `True` (limit hit)
4. Input: `circuit_breaker_state(5, 5)` → `"open"` (threshold)

---

7) Title: String Normalization & Tokenization

Overview: Normalize Unicode, strip punctuation, and tokenize complex strings.

Functionalities to be designed

Functionality	Input	Output	Description	Constraints
`normalize_whitespace`	`str`	`str`	Collapse spaces/tabs to single space; trim	Preserve word order
`casefold_str`	`str`	`str`	Unicode-aware case folding	`.casefold()`
`strip_ascii_punct`	`str`	`str`	Remove ASCII punctuation except spaces	Do not touch alphanumerics
`normalize_unicode_nfc`	`str`	`str`	Convert to NFC form	Requires `unicodedata`
`collapse_duplicates`	`str`, char `str`	`str`	Collapse consecutive duplicates of `char`	`char` length = 1
`tokenize_with_quotes`	`str`	`list[str]`	Split by spaces but preserve quoted substrings	Quotes `"..."` only

Test cases

1. Input: `normalize_whitespace(" a\t b c ")` → `"a b c"`
2. Input: `casefold_str("Straße")` → `"strasse"`
3. Input: `collapse_duplicates("a--b---c", "-")` → `"a-b-c"`
4. Input: `tokenize_with_quotes('say "hello world" now')` → `["say","hello world","now"]`

---

8) Title: String Parsing: Advanced Key=Value

Overview: Parse configurations with casting, defaults, multi-values, and stable formatting.

Functionalities to be designed

Functionality	Input	Output	Description	Constraints	
`parse_kv_pairs`	`str`	`dict[str,str]`	Parse `k=v` pairs separated by commas	Trim spaces; ignore empties	

Functionality	Input	Output	Description	Constraints	
cast_by_schema	dict[str,str], schema dict[str,Callable]	`dict[str,Any]	None`	Cast values per schema	Fail if any cast fails
get_int_or_default	dict, key str, default int	int	Return int value or default	Invalid ints → default	
parse_multi_values	str	dict[str,list[str]]	Support `k=v1	v2` multi-values	`
format_kv_stable	dict[str,str]	str	Sort keys; join k=v by commas	Keys sorted ascending	
diff_configs	dict[str,str], dict[str,str]	dict[str,set[str]]	Keys only in left/right/both	"only_left","only_right","both"	

Test cases

1. Input: parse_kv_pairs("a=1, b=2") → {"a":"1","b":"2"}
2. Input: cast_by_schema({"a":"1","b":"3.5"}, {"a":int,"b":float}) → {"a":1,"b":3.5}
3. Input: parse_multi_values("x=a|b, y=c") → {"x":["a","b"],"y":["c"]}
4. Input: format_kv_stable({"b":"2","a":"1"}) → "a=1,b=2"

---

9) Title: List Utilities: Advanced Ordering & Set-Like Ops

Overview: Deduplicate stably, sort with keys, group, partition, rotate, and set-like operations preserving order.

Functionalities to be designed

Functionality	Input	Output	Description	Constraints
dedup_preserve_order	list[Any]	list[Any]	Remove duplicates, keep first	Hashable items only
sort_by_key_desc	list[Any], key Callable	list[Any]	Sort descending by key	Stable not required
group_by_key	list[Any], key Callable	dict[Any,list[Any]]	Group elements by key	Keys hashable
partition_by_predicate	list[Any], pred Callable	tuple[list[Any],list[Any]]	Split into (true_list,false_list)	Preserve order
rotate_right	list[Any], k int	list[Any]	Rotate right by k	k mod n applied
ordered_union	list[Any], list[Any]	list[Any]	Union preserving first occurrence	Hashable items only

Test cases

1. Input: dedup_preserve_order([1,2,1,3]) → [1,2,3]
2. Input: group_by_key(["aa","b","ccc"], len) → {2:["aa"],1:["b"],3:["ccc"]}
3. Input: partition_by_predicate([1,2,3,4], lambda x: x%2==0) → ([2,4],[1,3])
4. Input: rotate_right([1,2,3,4], 1) → [4,1,2,3]

---

10) Title: List Analytics: Statistics & Outliers

Overview: Provide basic and extended statistics including quantiles and IQR-based outlier detection.

Functionalities to be designed

Functionality	Input	Output	Description	Constraints	
safe_min	list[float]	`float	None`	Minimum or None if empty	—
safe_max	list[float]	`float	None`	Maximum or None if empty	—
mean	list[float]	`float	None`	Average or None if empty	Sum/len

Functionality	Input	Output	Description	Constraints	
`median`	`list[float]`	`float	None`	Middle value or average of two	Sorted copy
`quantile`	`list[float]`, q `float`	`float	None`	q ∈ [0,1], linear interpolation	Empty → `None`
`iqr_outliers`	`list[float]`	`list[float]`	Values outside [Q1−1.5*IQR, Q3+1.5*IQR]	Return outlier values	

Test cases

1. Input: `median([1,3,2])` → `2.0` (sorted middle)
2. Input: `quantile([1,2,3,4], 0.25)` → `1.75` (linear interpolation)
3. Input: `iqr_outliers([1,2,3,100])` → `[100]` (high outlier)
4. Input: `mean([])` → `None` (empty)

---

11) Title: List Comprehensions: Multi-Stage Transforms

Overview: Use comprehensions to apply conditional transforms and conversions.

Functionalities to be designed

Functionality	Input	Output	Description	Constraints
`squares_of_evens`	`list[int]`	`list[int]`	Square even numbers	Preserve order
`cubes_of_odds`	`list[int]`	`list[int]`	Cube odd numbers	Preserve order
`strip_and_upper_all`	`list[str]`	`list[str]`	Trim and uppercase; drop empties	—
`flatten_2d`	`list[list[Any]]`	`list[Any]`	Flatten 2D list	Exactly two levels
`to_string_only_ints`	`list[Any]`	`list[str]`	Convert only ints to strings	Filter non-ints
`dict_from_pairs`	`list[tuple[str,int]]`	`dict[str,int]`	Build dict from pairs	Last occurrence wins

Test cases

1. Input: `squares_of_evens([1,2,3,4])` → `[4,16]`
2. Input: `strip_and_upper_all([" a ","","B "])` → `["A","B"]`
3. Input: `flatten_2d([[1],[2,3]])` → `[1,2,3]`
4. Input: `dict_from_pairs([("a",1),("a",2)])` → `{"a":2}`

---

12) Title: Nested Lists & Matrix Operations

Overview: Work with nested lists to flatten, transpose, and multiply matrices (conceptual constraints).

Functionalities to be designed

Functionality	Input	Output	Description	Constraints
`flatten_n_levels`	nested list, levels `int`	`list[Any]`	Flatten up to `levels`	levels ≥ 1
`transpose_matrix`	`list[list[Any]]`	`list[list[Any]]`	Swap rows/columns	Rectangular matrix
`matrix_add`	`list[list[float]]`, `list[list[float]]`	`list[list[float]]`	Element-wise add	Same shape
`matrix_scalar_mul`	`list[list[float]]`, scalar `float`	`list[list[float]]`	Multiply each element	—
`matrix_row_sums`	`list[list[float]]`	`list[float]`	Sum per row	—
`matrix_mul`	`list[list[float]]`, `list[list[float]]`	`list[list[float]]`	Standard matrix multiply	Inner dims compatible

Test cases

1. Input: `transpose_matrix([[1,2],[3,4]])` → `[[1,3],[2,4]]`
2. Input: `matrix_add([[1],[2]], [[3],[4]])` → `[[4],[6]]`
3. Input: `matrix_scalar_mul([[1,2]], 2.0)` → `[[2.0,4.0]]`
4. Input: `matrix_row_sums([[1,2],[3,4]])` → `[3,7]`

---

13) Title: Dictionary Frequency: N-grams & Probabilities

Overview: Count frequencies, top-k with ties, normalize to probabilities, and prune rare items.

Functionalities to be designed

Functionality	Input	Output	Description	Constraints
`count_unigrams`	`list[str]`	`dict[str,int]`	Count tokens	Case-sensitive
`count_ngrams`	`list[str]`, n `int`	`dict[tuple[str,...],int]`	Count n-grams	n ≥ 1
`top_k_by_freq`	`dict[Any,int]`, k `int`	`list[tuple[Any,int]]`	Top k by freq; ties by key	Key sort for ties
`normalize_counts`	`dict[Any,int]`	`dict[Any,float]`	Convert to probabilities	Sum > 0
`merge_counts_weighted`	`dict[Any,int]`, `dict[Any,int]`, `float` w1, `float` w2	`dict[Any,float]`	Weighted merge	Non-negative weights
`prune_below_threshold`	`dict[Any,int]`, t `int`	`dict[Any,int]`	Remove entries with count < t	t ≥ 1

Test cases

1. Input: `count_ngrams(["a","b","a","b"], 2)` → `{("a","b"):2,("b","a"):1}`
2. Input: `normalize_counts({"x":2,"y":2})` → `{"x":0.5,"y":0.5}`
3. Input: `top_k_by_freq({"a":2,"b":2}, 1)` → `[("a",2)]` (tie broken by key)
4. Input: `prune_below_threshold({"a":1,"b":3}, 2)` → `{"b":3}`

---

14) Title: Dictionary: Safe Access & Deep Updates

Overview: Work with nested dictionaries using safe path operations and deep merges.

Functionalities to be designed

Functionality	Input	Output	Description	Constraints	
`get_nested`	dict, path `list[str]`	\`Any	None\`	Safe path retrieval	Return `None` if missing
`get_nested_default`	dict, path, default	`Any`	Return value or default	—	
`setdefault_path`	dict, path `list[str]`, default leaf	`dict`	Create path if missing	Mutates and returns dict	
`deep_merge_left_wins`	left dict, right dict	`dict`	Merge recursively; left overrides	Shallow for non-dicts	
`delete_path_safe`	dict, path	`bool`	Delete if exists; return success	—	
`path_exists`	dict, path	`bool`	Check if path exists	—	

Test cases

1. Input: `get_nested({"a":{"b":1}}, ["a","b"])` → `1`
2. Input: `setdefault_path({}, ["a","b","c"], 0)` → `{"a":{"b":{"c":0}}}`
3. Input: `deep_merge_left_wins({"a":{"x":1}}, {"a":{"x":2,"y":3}})` → `{"a":{"x":1,"y":3}}`
4. Input: `delete_path_safe({"a":{"b":1}}, ["a","b"])` → `True`

---

15) Title: Dict Comprehension: Advanced Transforms & Filters

Overview: Transform values with type-guards, filter keys via regex, invert with conflict strategy, renumber by sort.

Functionalities to be designed

Functionality	Input	Output	Description	Constraints
`square_int_values`	`dict[str,Any]`	`dict[str,int]`	Square only `int` values	Non-int values skipped
`filter_keys_regex`	`dict[str,Any]`, pattern `str`	`dict[str,Any]`	Keep keys matching regex	Use `re`
`invert_with_last_wins`	`dict[str,Any]`	`dict[Any,str]`	Invert; later keys overwrite	Values hashable

Functionality	Input	Output	Description	Constraints
`renumber_values_by_key_sort`	`dict[str,Any]`	`dict[str,int]`	Assign ranks by sorted keys	Ranks start at 1
`map_keys_to_case`	`dict[str,Any]`, mode `str`	`dict[str,Any]`	`'upper'`/`'lower'` key case	Preserve values
`freeze_dict_items`	`dict[str,Any]`	`tuple[tuple[str,Any],...]`	Freeze to tuple of items	Items sorted by key

Test cases

1. Input: `square_int_values({"a":2,"b":"x"})` → `{"a":4}`
2. Input: `filter_keys_regex({"foo":1,"bar":2}, r"^f")` → `{"foo":1}`
3. Input: `invert_with_last_wins({"a":1,"b":1})` → `{1:"b"}`
4. Input: `renumber_values_by_key_sort({"b":9,"a":8})` → `{"a":1,"b":2}`

---

16) Title: Indexing & Grouping with Custom Keys

Overview: Build index maps and groups using arbitrary key functions and produce derived views.

Functionalities to be designed

Functionality	Input	Output	Description	Constraints	
`index_by_func`	`list[Any]`, key `Callable`	`dict[Any,int]`	Map key(element) → index	Last occurrence wins	
`group_by_func`	`list[Any]`, key `Callable`	`dict[Any,list[Any]]`	Group elements by key	Keys hashable	
`group_to_sets`	dict groups	`dict[Any,set[Any]]`	Convert group lists to sets	—	
`invert_groups`	dict groups	`dict[Any,Any]`	Map element → its group key	Last group wins	
`count_by_predicate`	`list[int]`, pred	`dict[str,int]`	Counts `"true"`/`"false"` outcomes	Keys fixed	
`top_group_key`	dict groups	`Any	None`	Group key with max size	Ties: min key wins

Test cases

1. Input: `index_by_func(["a","bb","ccc"], len)` → `{1:0,2:1,3:2}`
2. Input: `group_by_func(["apple","angle","banana"], lambda s: s[0])` → `{"a":["apple","angle"],"b":["banana"]}`
3. Input: `group_to_sets({"x":[1,1,2]})` → `{"x":{1,2}}`
4. Input: `top_group_key({"a":[1,2], "b":[3]})` → `"a"`

---

17) Title: Functional Programming Pipelines (Map/Filter/Reduce)

Overview: Compose transformations, apply multiple predicates, and support lazy-like pipeline concepts.

Functionalities to be designed

Functionality	Input	Output	Description	Constraints
`double_all_map`	`list[int]`	`list[int]`	Double via `map`	—
`filter_all_predicates`	`list[int]`, preds `list[Callable]`	`list[int]`	Keep elements passing all preds	Order preserved
`filter_any_predicate`	`list[int]`, preds `list[Callable]`	`list[int]`	Keep elements passing any pred	—
`scan_cumulative_sum`	`list[int]`	`list[int]`	Running totals (prefix sums)	—
`compose_functions`	`list[Callable[[Any],Any]]`	`Callable[[Any],Any]`	Compose left-to-right	Identity for empty
`take_drop_while`	`list[int]`, pred	`tuple[list[int],list[int]]`	`(take_while, drop_while)`	—

Test cases

1. Input: filter_all_predicates([1,2,3,4], [lambda x: x>1, lambda x: x%2==0]) → [2,4]
2. Input: filter_any_predicate([1,2,3], [lambda x: x%2==0, lambda x: x>2]) → [2,3]
3. Input: scan_cumulative_sum([1,2,3]) → [1,3,6]
4. Input: take_drop_while([1,2,3,0,4], lambda x: x>0) → ([1,2,3], [0,4])

18) Title: Reduce Aggregations: Max/Ties/Concat/Escape

Overview: Use reduce-like semantics to aggregate with tie-breaking, concatenation, and scans.

Functionalities to be designed

Functionality	Input	Output	Description	Constraints	
product_or_none	list[int]	`int	None`	Product or None if empty	Identity 1
concat_with_sep_escaped	list[str], sep str, esc str	str	Escape sep within tokens	Empty list → ""	
max_by_key_tiebreak	list[Any], key, tiebreak Callable	`Any	None`	Max by key; tie via tiebreak	Empty → None
reduce_to_tree_pairs	list[Any]	list[tuple[Any,Any]]	Pair neighbors: (a0,a1), (a2,a3)...	Odd: last dropped	
scan_apply	list[Any], fn Callable	list[Any]	Apply cumulative function like scan	Deterministic	
reduce_with_initial	list[Any], fn Callable, init Any	Any	Reduce with initial value	Empty returns init	

Test cases

1. Input: product_or_none([2,3,4]) → 24
2. Input: product_or_none([]) → None
3. Input: concat_with_sep_escaped(["a","b|c"], "|", "\\") → "a|b\|c"
4. Input: reduce_to_tree_pairs([1,2,3]) → [(1,2)]

19) Title: OOP: Bank Account Model (Extended)

Overview: Design an Account with operations for transfers, history, overdraft, interest, and serialization.

Functionalities to be designed

Functionality	Input	Output	Description	Constraints
Account.__init__	owner str, initial float, overdraft float	Account	Initialize with limits	initial ≥ 0; overdraft ≥ 0
deposit	amount float	None	Increase balance	amount > 0
withdraw	amount float	bool	Deduct if ≥ -overdraft	Return success
transfer_to	target Account, amount float	bool	Move funds if possible	Atomic success/failure
apply_interest	rate float	None	Multiply balance by (1+rate)	rate ≥ 0
to_dict	—	dict[str,Any]	Serialize owner, balance, overdraft	No history included

Test cases

1. Input: acc = Account("Alice", 100.0, 50.0); acc.withdraw(120.0) → True, balance -20.0
2. Input: acc.transfer_to(Account("Bob", 0.0, 0.0), 50.0) → True (funds moved)
3. Input: acc.apply_interest(0.1) with balance 100.0 → balance 110.0
4. Input: acc.to_dict() → {"owner":"Alice","balance":..., "overdraft":...}

20) Title: Regex: Log Parsing & Validation (Advanced)

Overview: Use regular expressions for extracting IPs, timestamps, UUIDs, log metadata, and redacting PII.

Functionalities to be designed

Functionality	Input	Output	Description	Constraints			

Functionality	Input	Output	Description	Constraints			
`extract_ipv4`	`str`	`str	None`	First IPv4 match	Basic pattern; return `None` if absent		
`extract_ipv6`	`str`	`str	None`	First IPv6 match	Simplified compressed forms		
`find_iso_timestamps`	`str`	`list[str]`	`YYYY-MM-DDThh:mm:ss` 24-hour	Zero-padded			
`extract_log_level_named`	`str`	`str	None`	Named group for `INFO	WARN	ERROR`	Case-sensitive
`is_valid_uuid4`	`str`	`bool`	Validate UUID v4 format	Hex + version `4`			
`redact_pii`	`str`	`str`	Replace emails/phones with placeholders	`"***@***"`, `"***-***-****"`			

Test cases

1. Input: `extract_ipv4("Client 192.168.0.1 connected")` → `"192.168.0.1"`
2. Input: `find_iso_timestamps("at 2025-12-20T09:15:00")` → `["2025-12-20T09:15:00"]`
3. Input: `is_valid_uuid4("123e4567-e89b-12d3-a456-426614174000")` → `True`
4. Input: `redact_pii("mail a@b.com, phone +1-202-555-0143")` → `"mail ***@***, phone ***-***-****"`

---

21) Title: I/O: CSV Summarizer (Extended)

Overview: Parse CSV-like strings, compute column stats, and format outputs.

Functionalities to be designed

Functionality	Input	Output	Description	Constraints	
`parse_csv_line`	`str`	`list[str]`	Split by commas; trim spaces	No quotes handling	
`sum_numeric_columns`	`list[list[str]]`	`list[float]`	Sum columns, cast to float	Non-numeric → 0.0	
`avg_numeric_columns`	`list[list[str]]`	`list[float]`	Average per column	Empty rows ignored	
`min_numeric_columns`	`list[list[str]]`	`list[float	None]`	Min per column	All non-numeric → `None`
`max_numeric_columns`	`list[list[str]]`	`list[float	None]`	Max per column	All non-numeric → `None`
`format_csv_row`	`list[Any]`	`str`	Join with commas	Values stringified	

Test cases

1. Input: `parse_csv_line(" 1, 2 ,x ")` → `["1","2","x"]`
2. Input: `sum_numeric_columns([["1","2"],["x","3"]])` → `[1.0,5.0]`
3. Input: `avg_numeric_columns([["1","2"],["3","4"]])` → `[2.0,3.0]`
4. Input: `min_numeric_columns([["x","y"]])` → `[None, None]`

---

22) Title: Types: Tuple vs List APIs (Extended)

Overview: Demonstrate mutability differences and safe transformations between sequences.

Functionalities to be designed

Functionality	Input	Output	Description	Constraints
`append_to_list_copy`	`list[int]`, x `int`	`list[int]`	Return new list with x	Original not mutated
`extend_tuple`	`tuple[int,...]`, x `int`	`tuple[int,...]`	Return new tuple with x	Create new tuple
`concat_list_tuple`	`list[int]`, `tuple[int,...]`	`list[int]`	Concatenate into list	—
`is_mutable_sequence`	`Any`	`bool`	True for list; False for tuple	Only list/tuple considered
`list_to_tuple_copy`	`list[Any]`	`tuple[Any,...]`	Copy list to tuple	Shallow copy

Functionality	Input	Output	Description	Constraints
`tuple_to_list_copy`	`tuple[Any,...]`	`list[Any]`	Copy tuple to list	Shallow copy

Test cases

1. Input: `append_to_list_copy([1,2], 3)` → `[1,2,3]`
2. Input: `extend_tuple((1,2), 3)` → `(1,2,3)`
3. Input: `is_mutable_sequence((1,))` → `False`
4. Input: `concat_list_tuple([1], (2,3))` → `[1,2,3]`

---

23) Title: Control Flow: Exceptions, Defaults & Fallbacks

Overview: Parse with exception safety, divide with guards, and chain fallbacks lazily.

Functionalities to be designed

Functionality	Input	Output	Description	Constraints	
`parse_int_or_default`	`str`, default `int`	`int`	`int(...)` or default	Catch `ValueError`	
`safe_divide`	`float` a, `float` b	\`float	None\`	`a/b` or `None` if `b==0`	No exception
`first_valid`	`list[Callable[[],Any]]`	\`Any	None\`	First non-`None` result	Lazy evaluation
`retry_on_exception`	fn `Callable`, attempts `int`	\`Any	None\`	Try until success or exhaust	Exceptions caught
`default_if_none`	`Any`, default `Any`	`Any`	Return value or default	—	
`try_parse_float_chain`	`list[str]`	\`float	None\`	First string that parses to float	Left-to-right

Test cases

1. Input: `parse_int_or_default("x", 5)` → `5`
2. Input: `safe_divide(10.0, 0.0)` → `None`
3. Input: `first_valid([lambda: None, lambda: 3])` → `3`
4. Input: `default_if_none(None, 7)` → `7`

---

24) Title: Strings: Multi-Delimiter Tokenizer & Cleaner

Overview: Split on multiple delimiters, clean tokens, and re-join consistently.

Functionalities to be designed

Functionality	Input	Output	Description	Constraints
`split_on_any`	`str`, delims `set[str]`	`list[str]`	Split on any delimiter	Remove empty tokens
`trim_tokens`	`list[str]`	`list[str]`	Strip spaces around each token	—
`lower_nonempty_tokens`	`list[str]`	`list[str]`	Lowercase and drop empties	—
`join_with_space`	`list[str]`	`str`	Join tokens with single space	No leading/trailing space
`count_unique_tokens`	`list[str]`	`int`	Number of unique tokens	Case-sensitive
`replace_tokens`	`list[str]`, map `dict[str,str]`	`list[str]`	Replace via mapping	Unmapped tokens unchanged

Test cases

1. Input: `split_on_any("a,b;c|d", {",",";","|"})` → `["a","b","c","d"]`
2. Input: `lower_nonempty_tokens([" A ","","B"])` → `["a","b"]`
3. Input: `join_with_space(["a","b","c"])` → `"a b c"`
4. Input: `replace_tokens(["x","y"], {"x":"X"})` → `["X","y"]`

---

25) Title: Lists: Sliding Window, Differences & Normalization

Overview: Provide windowed sums, adjacent differences, normalization, and robustness to edge cases.

Functionalities to be designed

Functionality	Input	Output	Description	Constraints
sliding_window_sum	list[int], k int	list[int]	Sum for each window size k	k > 0; k ≤ len ⇒ windows else []
adjacent_diff	list[int]	list[int]	a[i+1] - a[i]	len n-1
normalize_min_max	list[float]	list[float]	Scale to [0,1]	All equal → zeros
zscore_normalize	list[float]	list[float]	(x - mean)/std	std == 0 → zeros
window_max	list[int], k int	list[int]	Max per window	k > 0
pad_to_length	list[Any], n int, pad Any	list[Any]	Right-pad list to length n	If len ≥ n, return copy

Test cases

1. Input: sliding_window_sum([1,2,3,4], 2) → [3,5,7]
2. Input: adjacent_diff([3,7,2]) → [4,-5]
3. Input: normalize_min_max([2,2,2]) → [0.0,0.0,0.0]
4. Input: pad_to_length([1,2], 4, 0) → [1,2,0,0]

---

26) Title: List Comprehension: Cartesian Products & Limits

Overview: Generate pairs with arithmetic constraints and unique summaries.

Functionalities to be designed

Functionality	Input	Output	Description	Constraints		
pairs_sum_even	list[int], list[int]	list[tuple[int,int]]	All (x,y) with even x+y	Preserve order		
pairs_product_within	list[int], list[int], limit int	list[tuple[int,int]]	(x,y) with x*y ≤ limit	Inclusive		
pairs_sum_unique	list[tuple[int,int]]	set[int]	Unique sums x+y	Use set		
pairs_diff_abs_lt	list[int], list[int], d int	list[tuple[int,int]]	Keep pairs with `	x-y	< d`	d ≥ 0
pairs_non_zero	list[int], list[int]	list[tuple[int,int]]	Keep pairs with non-zero product	—		
pairs_best_by_sum_top_k	pairs list, k int	list[tuple[int,int]]	Top k by x+y descending	Stable not required		

Test cases

1. Input: pairs_sum_even([1,2],[3,4]) → [(1,3),(2,4)]
2. Input: pairs_product_within([1,3],[2,4], 4) → [(1,2)]
3. Input: pairs_diff_abs_lt([1,5],[2,6], 2) → [(1,2),(5,6)]
4. Input: pairs_sum_unique([(1,3),(2,4)]) → {4,6}

---

27) Title: Dictionaries: Stable Merge, Precedence & Selection

Overview: Merge with precedence, partition keys, select subsets, and reconcile differences.

Functionalities to be designed

Functionality	Input	Output	Description	Constraints
merge_left_precedence	left dict, right dict	dict	Left overrides; otherwise take right	Shallow merge
diff_keys	left dict, right dict	dict[str,set[str]]	Partition keys	"only_left","only_right","both"
select_keys	dict, keys set[str]	dict	Subset of dict	Missing ignored
merge_right_precedence	left dict, right dict	dict	Right overrides	Shallow merge
common_key_values	left dict, right dict	dict[str,tuple[Any,Any]]	Shared keys map to (left,right)	—

Functionality	Input	Output	Description	Constraints
`drop_keys`	dict, keys `set[str]`	`dict`	Remove provided keys	Nonexistent ignored

Test cases

1. Input: `merge_left_precedence({"a":1}, {"a":2,"b":3})` → `{"a":1,"b":3}`
2. Input: `diff_keys({"a":1,"b":2},{"b":3,"c":4})` → `{"only_left":{"a"},"only_right":{"c"},"both":{"b"}}`
3. Input: `select_keys({"x":1,"y":2}, {"x","z"})` → `{"x":1}`
4. Input: `common_key_values({"a":1},{"a":2,"b":3})` → `{"a":(1,2)}`

---

28) Title: Dict Comprehension: Index Maps & Defaults

Overview: Create index maps, reverse maps, and default lookups via comprehensions.

Functionalities to be designed

Functionality	Input	Output	Description	Constraints
`index_map`	`list[str]`	`dict[str,int]`	Map string to index	0-based
`index_or_minus_one`	index map, keys `list[str]`	`dict[str,int]`	Lookup index or -1	—
`reverse_index_map`	index map	`dict[int,str]`	Reverse mapping	Unique indices
`index_map_by_keyfunc`	`list[str]`, key `Callable`	`dict[str,int]`	Index of first element per key	First occurrence wins
`index_map_stable`	`list[str]`	`dict[str,int]`	Preserve earliest index per key	First occurrence wins
`merge_index_maps_left`	left map, right map	`dict[str,int]`	Left precedence for overlaps	—

Test cases

1. Input: `index_map(["a","b","c"])` → `{"a":0,"b":1,"c":2}`
2. Input: `index_or_minus_one({"a":0}, ["a","z"])` → `{"a":0,"z":-1}`
3. Input: `reverse_index_map({"x":1,"y":2})` → `{1:"x",2:"y"}`
4. Input: `index_map_stable(["a","b","a"])` → `{"a":0,"b":1}`

---

29) Title: OOP: Library Catalog (Extended)

Overview: Design `Book` and `Library` classes with richer behaviors and constraints.

Functionalities to be designed

Functionality	Input	Output	Description	Constraints	
`Book.__init__`	title `str`, author `str`, year `int`, isbn `str`	`Book`	Initialize book	year ≥ 0; non-empty strings	
`Library.add_book`	`Book`	`bool`	Add if not duplicate	Duplicate by `(title,author)`	
`Library.search_by_author`	`str`	`list[Book]`	Return matching books	Case-sensitive	
`Library.remove_by_title`	`str`	`bool`	Remove first by title	Return success	
`Library.find_by_isbn`	`str`	`Book	None`	Lookup by ISBN	Unique constraint
`Library.list_titles_sorted`	—	`list[str]`	Titles sorted ascending	Stable for duplicates removed	

Test cases

1. Input: `Library().add_book(Book("Dune","Herbert",1965,"X"))` → `True`
2. Input: `Library().search_by_author("Herbert")` → `[Book(...)]`
3. Input: `Library().remove_by_title("Dune")` → `True`
4. Input: `Library().find_by_isbn("X")` → `Book("Dune","Herbert",1965,"X")`

---

30) Title: Regex: Validation & Extraction (Extended)

Overview: Validate phones, extract domains, find hashtags/mentions, and split URLs.

Functionalities to be designed

Functionality	Input	Output	Description	Constraints	
`is_valid_international_phone`	`str`	`bool`	`^\+\d{6,15}$` digits	Simple E.164-like	
`extract_domain`	`str` URL	`str	None`	Domain part from `http/https`	No subdomain stripping
`find_hashtags`	`str`	`list[str]`	`#` followed by `\w+`	Alnum and underscore	
`find_mentions`	`str`	`list[str]`	`@` followed by `\w+`	Alnum and underscore	
`split_url_path`	`str` URL	`list[str]`	Path parts without query	`"/"` split	
`extract_query_params`	`str` URL	`dict[str,str]`	`key=value` pairs in query	No percent-decoding	

Test cases

1. Input: `is_valid_international_phone("+919876543210")` → `True`
2. Input: `extract_domain("https://example.com/path")` → `"example.com"`
3. Input: `find_hashtags("Use #Python3 and #data_science")` → `["#Python3","#data_science"]`
4. Input: `extract_query_params("https://x.com?a=1&b=2")` → `{"a":"1","b":"2"}`