**🐍 Python Assignment Functions & List Comprehensions**

**Topic 1 — Functions (def) & Lambda Functions**

**Part A — Basic Function Concepts (Single Return Values)**

1. Write a function wish(name) that returns "Hello, Good Morning <name>!".
2. Write a function cube(n) that returns the cube of a number (n³).
3. Write a function check\_even(n) that returns True if the number is even, else False.
4. Write a function greatest\_of\_two(a, b) that returns the larger of two numbers.
5. Write a function count\_letters(s) that returns the number of characters in a string.
6. Write a function power(value, exp=2) that returns the number raised to the power exp. (Default exponent = 2)

**Part B — Returning Multiple Values**

1. Write a function swap(a, b) that returns the two values swapped as a tuple.
2. Write a function add\_sub(a, b) that returns both sum and difference of two numbers.
3. Write a function circle\_stats(radius) that returns the **area** and **circumference** as a tuple.
4. Write a function min\_max\_sum(nums) that returns the **minimum**, **maximum**, and **sum** of a list.

**Part C — Working with Strings & Formatting**

1. Write a function concat(\*parts) that joins multiple strings together with spaces.
2. Write a function format\_name(first, last="") that returns "Last, First" if last name is provided, else "First".
3. Write a function greet\_user(first, last) that returns a message like "Welcome, First Last!".

**Part D — Lambda Functions**

1. Write a lambda function to add two numbers.
2. Write a lambda function mult = lambda \*args: that returns the product of all arguments.
3. Write a one-line lambda expression to flatten a list of pairs like [(a,b), (c,d)] into [a, b, c, d].

**Part E — Recursive, Higher-Order & Inner Functions**

1. Write a recursive function factorial(n) that returns the factorial of a number.
2. Write a function apply\_twice(func, x) that applies a function twice → func(func(x)).
3. Write a function inner\_demo() that defines and calls an **inner function** inside it.
4. Write a function call\_function(func, value) that takes another function as a parameter and applies it.
5. Write a function square\_list(nums) that defines a helper function square(x) inside it and returns a list of squares.

**Part F — Intermediate Function Concepts**

1. Write a function make\_multiplier(m) that returns a new function which multiplies input values by m.
2. Write a function filter\_primes(nums) that filters prime numbers using a helper is\_prime() function.
3. Write a function convert\_to\_fahrenheit(celsius\_list) using map and a lambda.  
   Formula: F = C \* 9/5 + 32
4. Write a function apply\_if(func, predicate, iterable) that applies a function only to items that pass a condition.
5. Write a function curry(func) that converts a two-argument function into a curried version → can be called as f(a)(b).
6. Write a function variadic\_sum(\*args, \*\*kwargs) that sums all positional arguments and multiplies by a scale factor if provided.
7. Write a function sort\_by\_length(strings) that sorts a list of strings by their length using a lambda as key.

**Topic 2 — List Comprehensions**

**Part A — Basic Comprehension Practice**

1. Create [1,2,3,4] → list of their **squares**.
2. Create ['a','b','c'] → list of their **uppercase** versions.
3. Make a list of **even numbers** from 1 to 20.
4. Given words = ['apple', 'bee', 'cat'], create [(word, len(word)) for word in words].
5. Flatten [[1,2], [3,4], [5]] → [1,2,3,4,5].
6. From nums = [0,1,2,3], create a list of 'even' or 'odd' strings.
7. Using a comprehension, make a list of booleans showing if numbers in a list are **prime** (use helper function).
8. From range(1,50), select numbers divisible by **both 2 and 3**.
9. Create coordinate pairs (i, j) where i and j go from 0 to 2.
10. From ['a b','c d e'], produce a flat list of all words.

**Part B — Using Lists of Dictionaries**

1. Given students = [{'name':'A','score':80}, {'name':'B','score':60}], Create a list of names of students with score ≥ 75.
2. Build a list of dictionaries like {'n': n, 'square': n\*n} for numbers 1 to 10.
3. Merge two lists element-wise into tuples, but include only pairs where **both elements are truthy**.
4. Create a comprehension to **transpose** a matrix (swap rows ↔ columns).

**PyPart C — Advanced Comprehensions**

1. Flatten a **3-level nested list** [[[1,2],[3]],[[4]]] into a single flat list.
2. Write a comprehension that keeps **only unique items** from a list while **preserving order**.
3. Find all **indices (positions)** where a given number appears in a list.
4. Generate all **palindromic numbers (as strings)** between 1 and 999.
5. Convert a dictionary like {'a':[1,2], 'b':[3]} into a list of (key, value) pairs.
6. Create a list of **prime pairs (p, q)** where both are primes under 100 and p + q is also prime.
7. Create a **lower-triangular matrix (n×n)** with 1s below and on the diagonal, 0s above.
8. Generate all possible **sequences of length 4** from 'abcd' with **no repeated letters**.