# DOCUMENTATION FOR PYTHON

## **PYTHON FUNCTIONS:**

1. **lambda functions:**

These are small anonymous functions defined using the `lambda` keyword.

add = lambda x, y: x + y

print(add(2, 3)) # Output: 5

1. **List comprehensions:**

squares = [x\*\*2 for x in range(10)]

print(squares) # Output: [0, 1, 4, 9, 16, 25, 36, 49, 64, 81]

1. **\*\*Dictionary comprehensions\*\*:**

square\_dict = {x: x\*\*2 for x in range(10)}

print(square\_dict) # Output: {0: 0, 1: 1, 2: 4, 3: 9, 4: 16, :5 25, 6: 36, 7: 49, 8: 64, 9: 81}

1. **map() function :**

Applies a function to all items in an input list.

nums = [1, 2, 3, 4, 5]

squares = list(map(lambda x: x\*\*2, nums))

print(squares) # Output: [1, 4, 9, 16, 25]

1. **filter() function:**

Filters items out of a list.

nums = [1, 2, 3, 4, 5, 6]

even\_nums = list(filter(lambda x: x % 2 == 0, nums))

print(even\_nums) # Output: [2, 4, 6]

1. **zip() function:**

Combines two or more lists into a list of tuples.

names = ['Alice', 'Bob', 'Charlie']

scores = [85, 92, 78]

combined = list(zip(names, scores))

print(combined) # Output: [('Alice', 85), ('Bob', 92), ('Charlie', 78)]

### **High Cohesion**

Cohesion refers to how closely related and focused the responsibilities of a single module or class are. High cohesion means that a module or class is designed with a single responsibility or a group of related responsibilities. This makes the code more understandable, easier to maintain, and less prone to errors.

**Benefits of High Cohesion:**

* Easier to understand and reason about.
* Easier to maintain and modify.
* Encourages reusability of modules/classes.
* Enhances readability and reduces complexity.

### **Low Coupling**

Coupling refers to the degree of direct knowledge that one module has of another. Low coupling means that modules or classes are designed to minimize dependencies on each other. This makes the system more modular, flexible, and easier to test.

**Benefits of Low Coupling:**

* Easier to modify and extend code.
* Improves testability of individual components.
* Reduces the impact of changes in one module on others.
* Enhances code maintainability and flexibility.