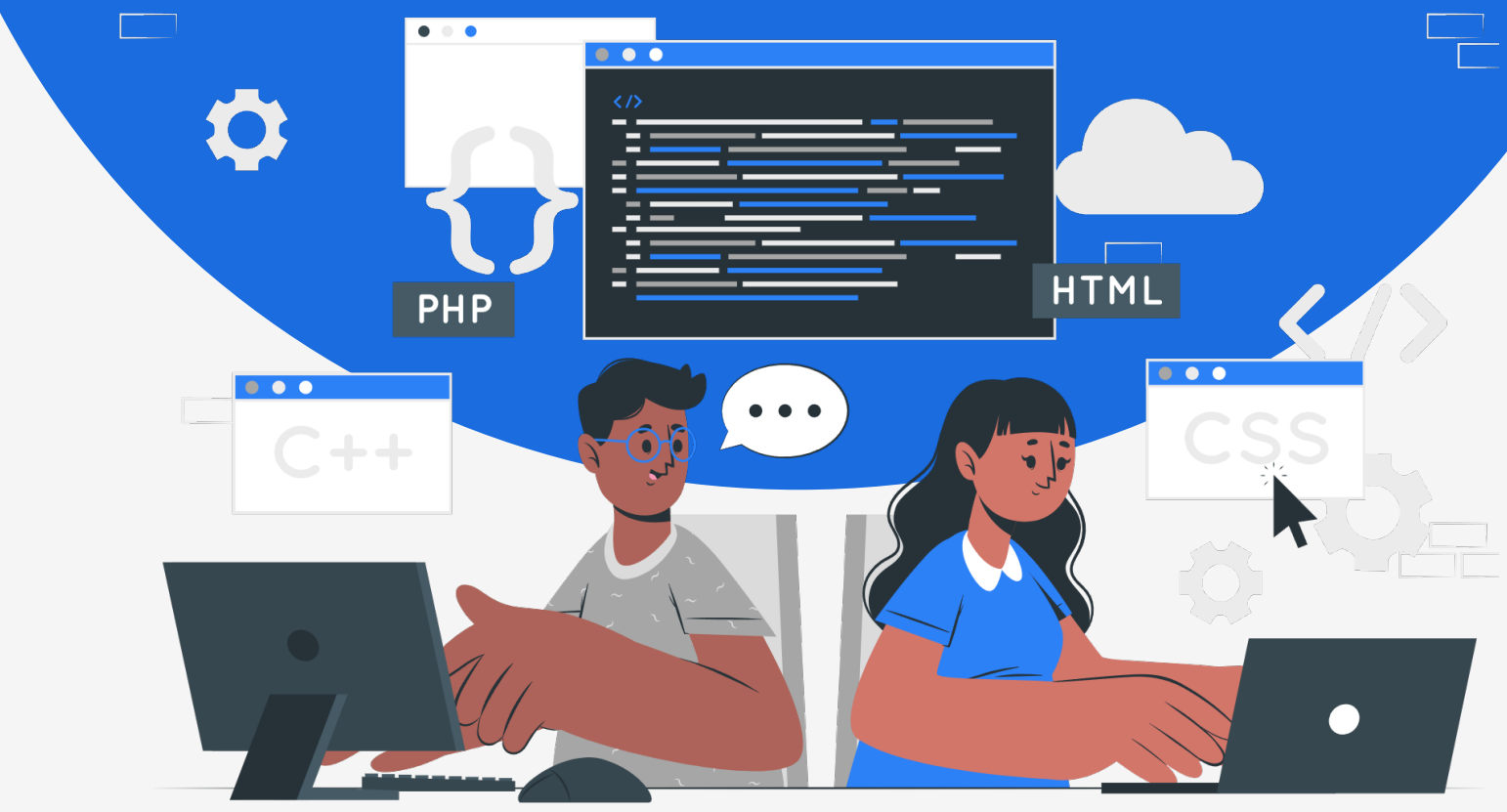
Lesson Plan

Map, Reduce, Filter Functions



PW SKILLS

Topics to be covered:

1. What is the map() function?
2. Working Process of map() function
3. map() function Examples
4. Analogy of map() function
5. What is the reduce() function?
6. Working Process of reduce() function
7. reduce() function Examples
8. Analogy of reduce() function
9. What is the filter() function?
10. Working Process of filter() function
11. filter() function Examples
12. Analogy of filter() function

1. What is the map() function?

- The `map()` function executes a specified function for each item in an `iterable`. The item is sent to the function as a parameter.
- The map() function in Python is a built-in function that applies a given function to all the items in an iterable (e.g., a list) and returns an iterable map object (an iterator) of the results.

The syntax for the map() function is:

```
map(function, iterable, ...)
```

2. Working Process of map() function:

Function Parameter:

- The first parameter of map() is the function that you want to apply to each item in the iterable.
- It can be a built-in function, a user-defined function, or a lambda function.

Iterable Parameters:

- The second parameter onward represents one or more iterables (e.g., lists, tuples, strings) on which the specified function will be applied.
- All iterables should have the same length. If one iterable is shorter than the others, map() stops when the shortest iterable is exhausted.

Iterating Over the Iterable(s):

- map() applies the specified function to each corresponding item in the iterables, moving in a parallel fashion.
- It starts by applying the function to the first items, then the second items, and so on.

Generating a Map Object (Iterator):

- map() returns an iterable map object. This is a special type of iterator that contains the results of applying the function to the items in the input iterables.

Converting Map Object to a Usable Form:

- To see the results, you need to convert the map object into a list, tuple, or another iterable form using the list(), tuple(), or other conversion functions.

3. map() function Examples:

1. Convert String Numbers to Integers:

```
numbers_as_strings = ["1", "2", "3", "4", "5"]
numbers_as_integers = list(map(int, numbers_as_strings))
print(numbers_as_integers)
# Output: [1, 2, 3, 4, 5]
```

2. Calculate Square of Each Number:

```
numbers = [1, 2, 3, 4, 5]
squared_numbers = list(map(lambda x: x**2, numbers))
print(squared_numbers)
# Output: [1, 4, 9, 16, 25]
```

3. Capitalize First Letter of Each Word:

```
words = ["python", "programming", "skills"]
capitalized_words = list(map(str.capitalize, words))
print(capitalized_words)
# Output: ['Python', 'Programming', 'Skills']
```

4. Calculate GPA from Grades:

```
grades = ["A", "B", "C", "A", "B"]
gpa = list(map(lambda grade: 4 if grade == "A" else (3
if grade == "B" else 2), grades))
print(gpa)
# Output: [4, 3, 2, 4, 3]
```

5. Combine First and Last Names:

```
first_names = ["John", "Alice", "Bob"]
last_names = ["Doe", "Smith", "Johnson"]
full_names = list(map(lambda first, last: f"{first}
{last}", first_names, last_names))
print(full_names)
# Output: ['John Doe', 'Alice Smith', 'Bob Johnson']
```

4. Analogy of map() function:

Let's consider an example of **A Factory Conveyor Belt:**

Imagine you are working in a factory where products move along a conveyor belt. Each product goes through a series of stations, and a specific operation is performed at each station. The **map()** function is like this conveyor belt, processing each item one by one.

- **Factory Stations (Function):**
- In our analogy, each station along the conveyor belt represents a function that needs to be applied to the products.
- Similarly, the function provided to map() is like the operation performed at each station.

- **Products on the Conveyor Belt (Iterable):**
- The products moving on the conveyor belt represent the elements of an iterable (e.g., a list).
- Just as the conveyor belt processes each product, map() processes each element of the iterable.

- **Operation at Each Station (Function Application):**
- As each product moves through a station, a specific operation is performed on it.
- Similarly, the function provided to map() is applied to each element of the iterable.

- **End Result (Iterable of Results):**
- At the end of the conveyor belt, you have a series of processed products.
- With map(), you get an iterable (map object) containing the results of applying the function to each element.

```
# Factory function: Function to double the value
def double(value):
    return value * 2

# List of products: Numbers to be processed
numbers = [1, 2, 3, 4, 5]


# Conveyor belt operation: Using map() to apply the
function to each number
result_map_object = map(double, numbers)

# Convert the map object to a list to see the results
result_list = list(result_map_object)

print(result_list)
# Output: [2, 4, 6, 8, 10]
```

5. What is the reduce() function?

Python's `reduce()` implements a mathematical technique commonly known as `folding` or `reduction`.

- It is a process where you reduce a list of items to a single value(cumulative value).
- Apply a function (or callable) to the first two items in an iterable and generate a partial result.
- Use that partial result, together with the third item in the iterable, to generate another partial result.
- Repeat the process until the iterable is exhausted and then return a single cumulative value.

The syntax for the reduce() function is:

```
functools.reduce(function, iterable[, initializer])
```

6. Working Process of reduce() function:

- **Function Parameter:**
- The first parameter of reduce() is the function that you want to apply cumulatively to the items of the iterable.
- The function should take two arguments and return a single result.

- **Iterable Parameter:**
- The second parameter is the iterable (e.g., a list) on which the specified function will be cumulatively applied.

- **Initializer (Optional):**
- The third, optional parameter is the initializer. If provided, it is placed before the items of the iterable in the calculation, and it serves as a default when the iterable is empty.

- **Iterative Application of Function:**
- The reduce() function applies the specified function cumulatively to the items of the iterable.
- It starts with the first two items, then applies the function to the result and the next item, and so on, until all items are processed.

- **Generating a Cumulative Result:**
- The result is a single cumulative value obtained by applying the function in a pairwise manner to the items of the iterable.

1. Calculate Sum of Numbers:

```
from functools import reduce
numbers = [1, 2, 3, 4, 5]
total_sum = reduce(lambda x, y: x + y, numbers)
print(total_sum)
# Output: 15
```

2. Multiply Elements of a List:

```
from functools import reduce
numbers = [2, 3, 4, 5]
product = reduce(lambda x, y: x * y, numbers)
print(product)
# Output: 120
```

3. Concatenate Strings:

```
from functools import reduce

words = ["Data", "Science", "Courses"]
concatenated_string = reduce(lambda x, y: x + " " + y,
words)
print(concatenated_string)
# Output: 'Data Science Courses'
```

4. Find Maximum Number in a List:

```
from functools import reduce

numbers = [12, 7, 22, 15, 8]
max_number = reduce(lambda x, y: x if x > y else y,
numbers)
print(max_number)
# Output: 22
```

5. Calculate Factorial of a Number:

```
from functools import reduce

def factorial(n):
    return reduce(lambda x, y: x * y, range(1, n + 1))

result = factorial(5)
print(result)
# Output: 120
```

7. Analogy of reduce() function:

Let's consider an example of **Shopping trip:**

Imagine you have a shopping list with various items, each priced differently. Your goal is to find the total cost of all the items on your list. The 'reduce()' function can be likened to the process of calculating the overall cost.

Here's how the analogy works:

1. Shopping List:

Think of your list of items as an iterable, where each item has a specific cost.

```
shopping_list = [
    {"item": "Apples", "cost": 1.50},
    {"item": "Bananas", "cost": 0.75},
    {"item": "Bread", "cost": 2.50},
    # ... (more items)
]
```

2. Mapping Function:

- Define a function that extracts the cost of each item.

```
def get_cost(item):
    return item["cost"]
```

3. Shopping Trip:

Now, you want to find the total cost of all the items on your shopping list.

```
from functools import reduce
total_cost = reduce(lambda x, y: x + y, map(get_cost,
shopping_list))
print(total_cost)


#output
4.75
```

In this analogy:

- **'map(get_cost, shopping_list)'** applies the get_cost function to each item, extracting the cost.
- **'reduce(lambda x, y: x + y, ...)'** adds up all the costs, reducing the iterable to a single value.

This is analogous to going through your entire shopping list, extracting the cost of each item, and then calculating the total cost of your entire shopping trip.

The reduce() function, in this context, mimics the process of accumulating or summarizing values from an iterable into a single result, which is similar to determining the total cost of items in your shopping cart.

Let's consider an example of Grocery Shopping with a Shopping List:

Imagine you're at a grocery store, and you have a shopping list that contains various items. The filter() function is like a smart shopping assistant who helps you pick only the items that meet specific criteria from your list.

- **Shopping List (Iterable):**
- Your shopping list represents an iterable (e.g., a list) containing various items (elements).

- **Shopping Assistant (Filter Function):**
- The filter() function acts as your shopping assistant. It takes your shopping list and a set of criteria to determine which items to pick.

- **Criteria for Selection (Filtering Function):**
- The criteria are defined by a filtering function that you provide to filter(). This function decides whether each item should be included in your final selection.

- **Item Selection (Filtering Process):**
- The filtering process involves going through each item on your shopping list and deciding whether to keep it or discard it based on the criteria.

- **Filtered Basket (Result):**
- The final result is a basket containing only the items that met the specified criteria.

```
# Shopping list: List of items to filter
shopping_list = ["Apples", "Bananas", "Milk", "Eggs",
"Bread"]

# Filtering function: Criteria for selection (e.g., only
items starting with 'B')
def starts_with_b(item):
    return item.startswith('B')

# Shopping assistant (filter function): Select items
based on the criteria
filtered_basket = filter(starts_with_b, shopping_list)

# Convert the filter object to a list to see the results
result_list = list(filtered_basket)

print(result_list)
# Output: ['Bananas', 'Bread']
```