**Week 1 Day 2 Task Description**

1. **RegEx**

Regular Expressions, often referred to as regex, are powerful pattern-matching tools used to search, extract, and manipulate text in Python. They provide a concise and flexible way to identify specific patterns within strings. First we need to import this to use in our code by using "import re"

Regular expressions offer a wide range of pattern-matching options, including character classes, anchors, modifiers, and more. They can be extremely powerful in various text processing tasks, such as data validation, parsing, and manipulation.The re module offers a set of functions that allows us to search a string for a match. And there are multiple Metacharcters, Special Sequences and Sets of these methods which can be used according to the requirements. Some of these are: '^' it is used when checking if the string starts with some specific keyword. '$' it is used when checking if the string ends with some specific keyword.

1. the re.search() method is used to search for a pattern within a string and return the first occurrence of a match. It scans the entire string and stops at the first match found.
2. The findall() method in Python's re module is used to find all non-overlapping occurrences of a pattern in a string and return them as a list of strings. It searches the entire input string, unlike search() that stops at the first match. It is commonly used when you need to extract multiple occurrences of a pattern from a string.
3. The split() method in Python is used to split a string into multiple substrings based on a specified delimiter. It returns a list of substrings obtained by splitting the original string. The delimiter determines where the split occurs, and it is excluded from the resulting substrings.
4. The sub() method in Python is used to substitute occurrences of a pattern in a string with a replacement value. It performs a search for the pattern in the string and replaces all occurrences with the provided replacement value.
5. A Match Object is an object containing information about the search and the result. The Match object has properties and methods used to retrieve information about the search, and the result:

.span() returns a tuple containing the start-, and end positions of the match. .string returns the string passed into the function .group() returns the part of the string where there was a match

### 2. Lambdas Functions

A lambda function is an anonymous function in Python defined using the lambda keyword. It allows you to create small, one-line functions without a name. Lambda functions are defined using the lambda keyword followed by the input arguments and a colon. They can take any number of arguments, but can only have a single expression. They are typically used when you need a small function for a specific task and don't want to define a separate named function. Lambda functions are often used in conjunction with higher-order functions like map(), filter(), and reduce().

### 3. List Comprehensions

List comprehensions in Python provide a concise way to create lists based on existing lists or other iterable objects. They allow you to combine loops and conditional statements into a single line of code, making it easier to create new lists with specific criteria. The basic syntax of a list comprehension is [expression for item in iterable if condition], where the expression defines the operation on each item, the item is the variable representing elements in the iterable, and the condition is an optional filtering condition.

### 4. Nested List Comprehensions

Nested list comprehensions refer to the usage of one or more list comprehensions within another list comprehension. It allows you to create complex nested structures or perform transformations on nested data.

**5. Decorators**

Decorators are a powerful tool in Python that allow programmers to modify the behavior of a function or class. They enable us to wrap a function, extending its behavior without permanently modifying it. This is achieved by using some key concepts such as functions as objects, nested functions, and closures. Understanding these concepts will provide a solid foundation for understanding and implementing decorators effectively.

### 6. Iterators

In Python, an iterator is an object that implements the iterator protocol, which consists of the iter() and next() methods. Iterators are used to iterate over a collection of elements or to generate a sequence of values on-the-fly.

The iter() method returns the iterator object itself. It is called when an iterator is initialized or reset. The next() method returns the next element in the sequence. It is called repeatedly until it raises the StopIteration exception, indicating that there are no more elements to be returned.

### 7. Yield Keyword

The yield keyword in Python is used in the context of defining generator functions. When a function contains the yield keyword, it becomes a generator function instead of a regular function. The yield keyword allows the function to produce a sequence of values that can be iterated over.

This allows for lazy evaluation of values, where the values are generated on-demand as they are requested, rather than generating the entire sequence upfront. It is memory-efficient and suitable for handling large or infinite sequences.

### 8. Generators Expression

Generator expressions in Python are similar to list comprehensions, but instead of generating a list, they create an iterator that produces the values on-the-fly. Generator expressions allow you to generate values dynamically without storing them in memory all at once.

The syntax for a generator expression is similar to that of a list comprehension, but with parentheses instead of square brackets. Instead of creating a list, it creates an iterator object that produces values when iterated over.

### 9. Paradigm

**Imperative Programming:** This paradigm focuses on describing the steps or instructions that the computer needs to follow to solve a problem. It is based on a sequence of statements that modify the program's state. Variables, loops, conditionals, and subroutines are commonly used in imperative programming.

**Object-Oriented Programming (OOP):** OOP is a programming paradigm that structures code around objects, which are instances of classes. It emphasizes encapsulation, inheritance, and polymorphism. OOP allows for organizing code into reusable and modular components, making it easier to manage and maintain complex systems.

**Functional Programming:** Functional programming treats computation as the evaluation of mathematical functions. It emphasizes immutability, pure functions, and higher-order functions. Functional programming avoids mutable state and focuses on composing functions and working with immutable data.

**Aspect-Oriented Programming (AOP):** AOP is a programming paradigm that aims to modularize cross-cutting concerns in a system. Cross-cutting concerns are features or functionalities that span multiple modules, such as logging, error handling, or security. AOP separates these concerns from the main codebase and allows for their modular and reusable implementation.