**Lists and methods**

A list is a data structure that can hold an ordered collection of items. These items can be of any data type, including integers, strings, floats, or even other lists. Lists are mutable, meaning that you can modify their elements after they have been created.

Here are some common methods that can be used with lists in Python: append(x), pop(), reverse(), sort(), count(x)

**Using Lists as Stacks and Queues**

A stack is a data structure that follows the Last-In, First-Out (LIFO) principle. Elements are added and removed from the same end, which is often referred to as the "top" of the stack. A queue is a data structure that follows the First-In, First-Out (FIFO) principle.

**List Comprehensions**

 List comprehensions are a concise and powerful way to create lists in Python. They allow us to generate a new list by applying an expression to each item in an existing iterable (like a list, tuple, or range) and optionally applying a condition.

**Nested List Comprehensions**

Nested list comprehensions in Python allow us to create lists that contain other lists. This can be useful for tasks like matrix operations or creating lists of lists with specific patterns. The syntax for a nested list comprehension is like a regular list comprehension, but it involves having one or more list comprehensions within another list comprehension.

**The del statement / Tuples and Sequences**

The del statement in Python is used to delete names (references) from the current namespace. A tuple is an immutable, ordered collection of elements. It can contain elements of any type, including other tuples. Tuples are created using parentheses (). Sequences in Python include not only tuples but also lists and strings. They share common sequence operations like indexing, slicing, and membership tests.

**Sets**

A set is an unordered collection of unique elements. It is a useful data structure when we want to store a collection of items without any specific order, and we want to ensure that each item is unique.

**Looping Techniques**

Looping is a fundamental concept in programming that allows us to iterate over a sequence of data or perform a specific task repeatedly. Python provides several looping techniques, including for loops, while loops, and list comprehensions.

**More on Conditions**

The comparison operators in and not in check whether a value occurs (does not occur) in a sequence. The operators is and is not compare whether two objects are really the same object; this only matters for mutable objects like lists.

**Comparing Sequences and Other Types**

In Python, we can compare different types of objects, including sequences like lists, tuples, and strings, with various operators. The comparison uses lexicographical ordering: first the first two items are compared, and if they differ this determines the outcome of the comparison; if they are equal, the next two items are compared, and so on, until either sequence is exhausted.

**Understanding errors and exceptions**

Errors and exceptions are a fundamental part of programming in Python. They represent situations where the interpreter encounters a problem while trying to execute a piece of code. Understanding and handling errors is essential for writing robust and reliable programs.

**Errors and Exceptions**

Handling errors and exceptions is an important part of writing robust and reliable code. It allows us to anticipate and gracefully handle unexpected situations, improving the overall resilience of your programs.

**Raising an Exception**

Raising an exception in Python means deliberately triggering an error or exceptional condition to signal that something unexpected or problematic has occurred in your code. This can be useful when we want to handle specific situations or indicate errors in our program. To raise an exception, we can use the "raise" statement followed by the type of exception you want to raise. You can also include an optional error message or additional information.

**Explaining Exceptions**

To raise an exception, we can use the raise statement followed by the type of exception we want to raise. We can also include an optional error message or additional information. Exception clauses are; try: except: else: and finally:

**Classes**

In Python, a class is a blueprint for creating objects. It defines the attributes (data members) and behaviours (methods) that the objects of the class will have. A class is a template or a blueprint, and objects as instances of that class. We define a class using a keyword "class" followed by a class name.

**Python Scopes and Namespaces**

In Python, a namespace is a mapping between names (identifiers) and objects. It allows you to uniquely identify objects in a program. Namespaces play a crucial role in determining the scope of a variable, i.e., where in the code you can access a particular variable or function. The scope of a variable refers to the region in a program where the variable is accessible. It determines which parts of the code can access a particular variable.

**Class and Instance Variables**

A class variable is a variable that is shared among all instances of a class. It is defined within the class but outside of any methods. Class variables are associated with the class itself, rather than with any specific instance. An instance variable is specific to each instance of a class. It is defined within the \_\_init\_\_ method using the self keyword. Each instance of the class has its own set of instance variables.

**Random Remarks**

Data attributes override method attributes with the same name; to avoid accidental name conflicts, which may cause hard-to-find bugs in large programs, it is wise to use some kind of convention that minimizes the chance of conflicts. Possible conventions include capitalizing method names, prefixing data attribute names with a small unique string (perhaps just an underscore), or using verbs for methods and nouns for data attributes.

**Daily Notes - Python Classes**

In Python, a class is a blueprint for creating objects. It provides a structure for defining the attributes (data members) and behaviours (methods) that the objects of the class will have. Classes allow us to model real-world entities or abstract concepts in our code. Once we have defined a class, we can create objects (instances) from it. Each object will have its own set of attributes and can perform the actions defined by the methods.