Week3 python

Lists

* append(x): Adds an item x to the end of the list.
* extend(iterable): Extends the list by appending all the items from the iterable to the end of the list.
* insert (i, x): Inserts an item x at a given position i. The first argument i is the index of the element before which to insert.
* remove(x): Removes the first item from the list whose value is equal to x. Raises a ValueError if there is no such item.
* pop([i]): Removes and returns the item at the given position i in the list. If no index is specified, pop () removes and returns the last item in the list.
* clear (): Removes all items from the list, effectively emptying it.
* index (x [, start [, end]]): Returns the index (position) in the list of the first item whose value is equal to x. Raises a ValueError if there is no such item. The optional start and end arguments limit the search to a specific subsequence of the list.
* count(x): Returns the number of times x appears in the list.
* sort (key=None, reverse=False): Sorts the items of the list in place. The key and reverse arguments can be used for customization. key can be a function that calculates a sort of key for each item, and reverse determines whether the sorting is in ascending (default) or descending order.
* reverse(): Reverses the order of elements in the list in place.
* copy (): Returns a shallow copy of the list. It's equivalent to a [:], where a is the original list. Changes to the copied list will not affect the original list, but changes to mutable objects within the list (e.g., lists or dictionaries) will be reflected in both the original and copied lists.

**Lists and stacks**

You can use lists to implement both stacks and queues, which are data structures used to manage collections of items with specific behavior.

* A stack is a data structure that follows the Last-In-First- Out (LIFO) principle, meaning the last item added to the stack is the first one to be removed.
* You can use the append() method to add items to the end of the list, and the pop() method to remove and return the last item.

Using a List as a Queue:

* A queue is a data structure that follows the First-In First-Out (FIFO) principle, meaning the first item added to the queue is the first one to be removed.
* While lists are not the most efficient data structure for queues, you can still use them by using the append() method to enqueue (add to the end) and the pop(0) method to dequeue (remove from the beginning)

Sets

* Uniqueness: Sets do not allow duplicate elements. If you try to add an element that already exists in the set, it won't be added again.
* Membership Testing: Sets are often used for checking membership, i.e., determining whether a particular element is present in the set.
* Creation: You can create sets using curly braces **{}** or by using the **set()** function. To create an empty set, you should use **set()**, as using **{}** would create an empty dictionary.
* Mathematical Operations: Sets support various mathematical operations, such as union, intersection, difference, and symmetric difference. These operations allow you to perform set operations like combining sets, finding common elements, or finding elements that are unique to each set.

Looping techniques

Looping is a powerful technique used to automate repetitive tasks, iterate through data structures, and control the flow of your program. The choice of loop depends on the specific requirements of your program and the data you need to work with.

Errors and Exceptions.

* **Error Messages**: When Python encounters a syntax error, it generates an error message to help you identify and fix the issue.
* **Error Location**: The error message points to the location in the code where the error was detected. It often includes an arrow (^) to indicate the specific character where the problem occurred.
* **Preceding Token**: The error message identifies the token preceding the arrow as the source of the error. In the given example, the missing colon (":") before the "print" keyword is the cause of the error.
* **File Name and Line Number**: The error message also provides the name of the file and the line number where the error occurred. This information helps you quickly locate and fix the issue in your code.

Exceptions: Exceptions are errors that occur during the execution of a Python program. These errors can occur even if the code is syntactically correct, and they are referred to as exceptions because they represent exceptional or unexpected situations.

Not Unconditionally Fatal: Unlike syntax errors, which prevent the program from running, exceptions are not unconditionally fatal. Python provides mechanisms to handle exceptions and gracefully recover from unexpected situations.

Handling Exceptions: Python allows you to write code that anticipates and handles exceptions. You can use try and except blocks to catch and respond to exceptions, allowing your program to continue running even when errors occur.

Error Messages: When an exception occurs and is not handled by the program, Python generates error messages that provide information about the type of exception and the context in which it occurred. These messages help developers identify and fix issues in their code.