

**“AZƏRBAYCAN HAVA YOLLARI” CJSC NATIONAL AVIATION ACADEMY**

**Individual Work № 3:**

**Topic: Lists. Sets. Stacks. Queues. Time complexity of different operations.**

**Subject: Obyektyönümlü proqramlaşdırma**

**Teacher: Mamed Shahmaliyev**

**Group: 1459i Student: Karimova Zakhra**

**Date: Signature: \_\_\_\_\_\_\_\_\_\_\_\_\_**

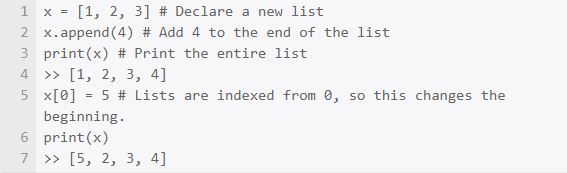
**­ Baku 2022**

**Lists**

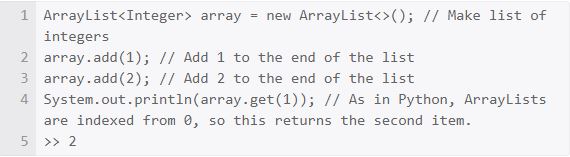
A list is a common data structure in most programming languages.

It functions as a collection of items. At a minimum, it usually provides the ability to add or remove items at the end of the list, and to look up items at a particular location in the list.

For example, in Python, you can create a list using square brackets.

****

In Java, the most commonly used list is the ArrayList in the standard library. As part of the declaration, you have to say what kind of data type you’re going to store in the list.



**LIST-INSERT(L,x)**

next[x] <- head[L]

if head[L] != NIL

then prev[head[L]] <- x

head[L] <- x

prev[x] <- NIL

Runtime? Θ(1) Deleting from a Linked List

**Remove an element x from list L**

• assume given pointer to x—we’ll “splice” out x

– how to generalize to deleting element given only key? use LIST-SEARCH function

LIST-DELETE(L,x)

if prev[x] != NIL

then next[pred[x]] <- next[x]

else head[L] <- next[x]

if next[x] != NIL

then pred[next[x]] <- prev[x]

**Sets**

A set is an unordered collection of items. Every set element is unique (no duplicates) and must be immutable (cannot be changed).

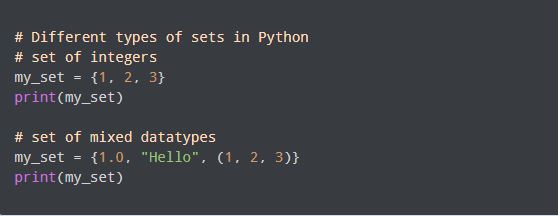
However, a set itself is mutable. We can add or remove items from it.

Sets can also be used to perform mathematical set operations like union, intersection, symmetric difference, etc.

## Creating Python Sets

A set is created by placing all the items (elements) inside curly braces {}, separated by comma, or by using the built-in set() function.

It can have any number of items and they may be of different types (integer, float, tuple, string etc.). But a set cannot have mutable elements like lists, sets or dictionaries as its elements.



**Output**

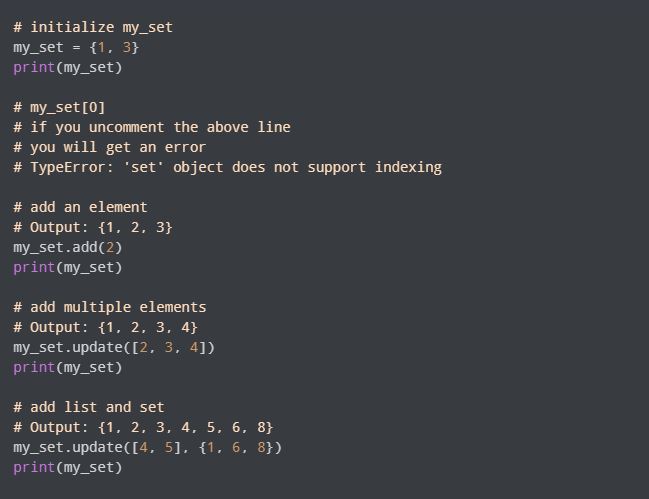
****

## Modifying a set in Python

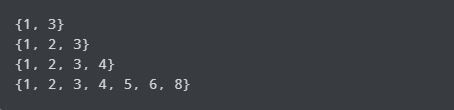
Sets are mutable. However, since they are unordered, indexing has no meaning.

We cannot access or change an element of a set using indexing or slicing. Set data type does not support it.

We can add a single element using the add() method, and multiple elements using the update() method. The update() method can take tuples, lists, strings or other sets as its argument. In all cases, duplicates are avoided.



**Output**

****

### **What is the Queue?**

A queue is a linear type of data structure used to store the data in a sequentially. The concept of queue is based on the FIFO, which means "**First in First Out**". It is also known as "first come first severed". The queue has the two ends front and rear. The next element is inserted from the **rear** end and removed from the **front** end.

**For example -** There are 20 computers in the computer science lab and connected to a single printer. The students want to print their paper; the printer will print the first task and second, so on. If we are the last in line, we need to wait until all other tasks are completed that ahead of ours.

The operating system manages the queue for processing the various processes within a computer.

Operations in Python

We can perform the following operations in the Queue.

* **Enqueue -** The enqueue is an operation where we add items to the queue. If the queue is full, it is a condition of the **Queue** The time complexity of enqueue is **O(1)**.
* **Dequeue -** The dequeue is an operation where we remove an element from the queue. An element is removed in the same order as it is inserted. If the queue is empty, it is a condition of the **Queue Underflow**. The time complexity of dequeue is **O(1)**.
* **Front -** An element is inserted in the front end. The time complexity of front is **O(1)**.
* **Rear -** An element is removed from the rear end.. The time complexity of the rear is **O(1)**.

## Methods Available in Queue

[Python](https://www.javatpoint.com/python-tutorial) provides the following methods, which are commonly used to perform the operation in Queue.

* **put(item) -** This function is used to insert element to the queue.
* **get() -** This function is used to extract the element from the queue.
* **empty() -** This function is used to check whether a queue is empty or not. It returns true if queue is empty.
* **qsize -** This function returns the length of the queue.
* **full() -** If the queue is full returns true; otherwise false.

**Time complexity of different operations**

What is the time complexity of operations?

So, the time complexity is the number of operations an algorithm performs to complete its task (considering that each operation takes the same amount of time). The algorithm that performs the task in the smallest number of operations is considered the most efficient one in terms of the time complexity. However, the space and time complexity are also affected by  factors such as your operating system and hardware, but we are not including them in this discussion.

Now to understand the time complexity, we will take an example in which we’ll compare two different algorithms which are used to solve a particular problem.

The problem is searching. We have to search for an element in an array (in this problem, we are going to assume that the array is sorted in  ascending order). To solve this problem we have two algorithms:

1. Linear Search.

2. Binary Search.

Let’s say the array contains ten elements, and we have to find the number ten in the array.



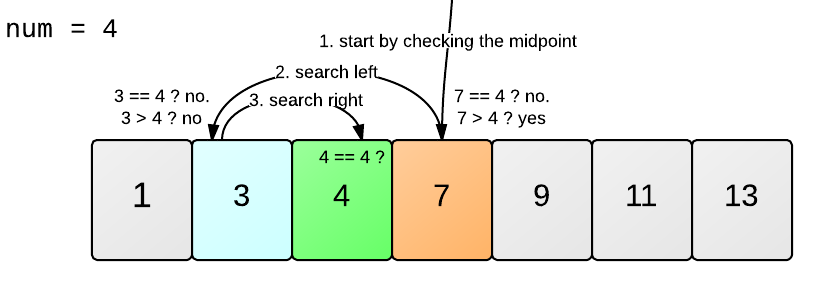
**Linear search** algorithm will compare each element of the array to the **search\_digit.** When it finds the **search\_digit** in the array, it will return **true**.

Now let’s count the number of operations it performs. Here, the answer is 10 (since it compares every element of the array). So,  Linear search uses ten operations to find the given element (these are the maximum number of operations for this array; in the case of Linear search, this is also known as the worst case of an algorithm).

In general, Linear search will take **n**number of operations in its worst case (where **n** is the size of the array).

Let’s examine the **Binary search** algorithm for this case.

Binary search can be easily understood by this example:



If we try to apply this logic on our problem then, first we’ll compare **search\_digit** with the middle element of the array, that is 5. Now since 5 is less than 10, then we will start looking for the **search\_digit** in the array elements greater than 5, in the same way until we get the desired element 10.

Now, try to count the number of operations binary search took to find the desired element. It took approximately four operations. Now, this was the worst case for binary search. This shows that there is a logarithmic relation between the number of operations performed and the total size of the array.

number of operations = log(10) = 4(approx)  
for base 2

We can generalize this result for Binary search as:

For an array of size **n**, the number of operations performed by the Binary Search is: **log(n)**