Python FSD

TECHNOLOGY

Stacks, Queues and Maps

simplilearn

Learning Objectives

By the end of this lesson, you will be able to:

- Understand Stack and Queue
- Compare Stack VS Queue
- Interpret Byte Array and Binary Tree

What is Stack ?

What is Stack

- Stack is a data structure that allows for last-in, first-out (LIFO) access. This means that the last element added to the stack is the first one to be removed.

- Python provides a built-in list data structure that can be used as a stack.

- To use a list as a stack, you can use the **append()** method to add elements to the top of the stack, and the **pop()** method to remove elements from the top of the stack.

What is Stack

Here's an example of how you can create a stack in Python:

```
# create an empty stack
        stack = []
# push elements onto the stack
        stack.append(1)
        stack.append(2)
        stack.append(3)
# pop elements off the stack
        top_element = stack.pop()
       second_element = stack.pop()
```

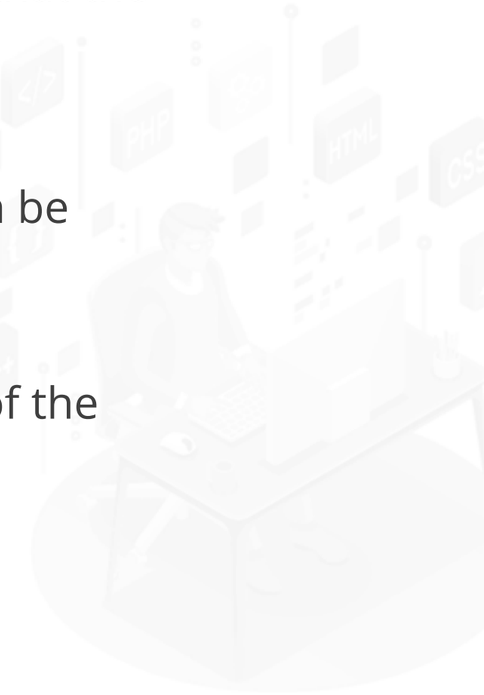
What is Queue

What is Queue

- Queue is a data structure that allows for first-in, first-out (FIFO) access. This means that the first element added to the queue is the first one to be removed.

- Python provides a built-in module called queue that includes a Queue class that can be used to create a queue.

- The operations performed on a queue are `enqueue` (add an element to the end of the queue) and `dequeue` (remove the first element from the queue).



What is Queue

Here's an example of how you can create a queue using the Queue class:

```
from queue import Queue
# create a new queue
        q = Queue()
# put elements into the queue
        q.put(1)
        q.put(2)
        q.put(3)
# get elements from the queue
    first_element = q.get()
    second_element = q.get()
```

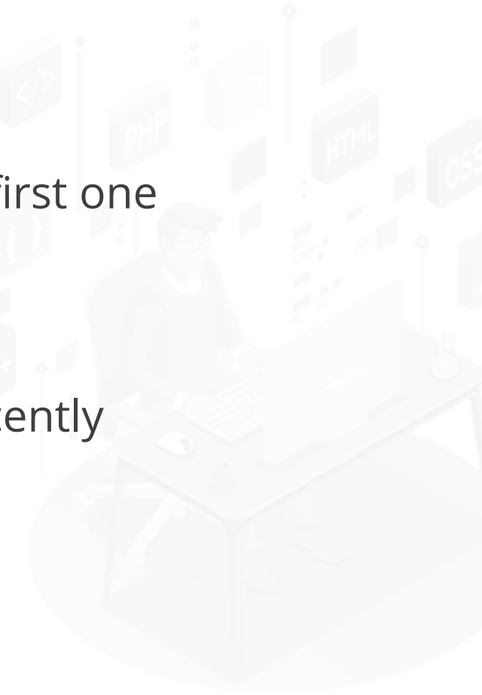
Types of Queue

Types of Queue

There are several types of queues available in the queue module. Here are some of the common types of queues:

Queue: This is a basic FIFO queue, where the first element added to the queue is the first one to be removed. This is the default type of queue in the queue module.

LifoQueue: This is a last-in, first-out (LIFO) queue, also known as a stack. The most recently added element is the first one to be removed.

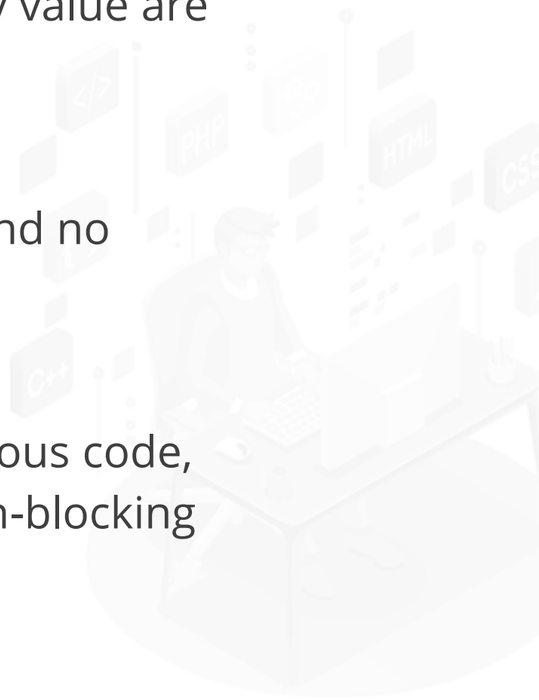


Types of Queue

PriorityQueue: This is a queue where each element is assigned a priority value, and elements are removed from the queue based on their priority. Elements with a higher priority value are removed first.

SimpleQueue: This is a simplified version of the Queue class, with fewer features and no support for blocking or timeouts.

SimpleAsyncQueue: This is an asynchronous queue designed for use in asynchronous code, such as with the asyncio module. It supports the async and awaits keywords for non-blocking usage.



Stack VS Queue

Stack VS Queue

Here are some differences between stack and queue:

Order of elements: In a stack, the last element added is the first one to be removed, whereas, in a queue, the first element added is the first one to be removed.

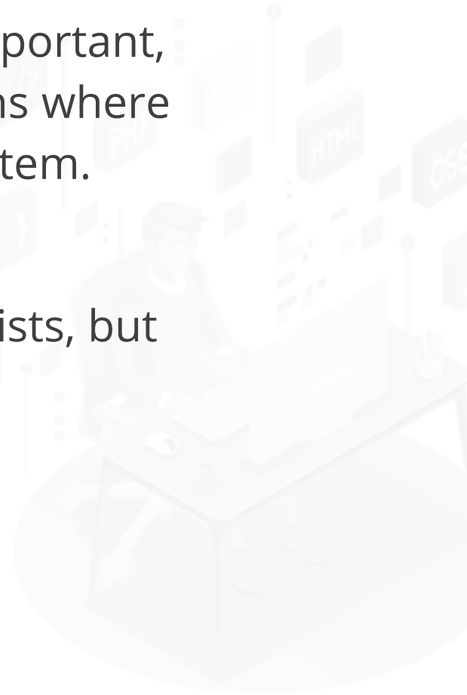
Operations: The operations performed on a stack are push and pop, while the operations performed on a queue are enqueue and dequeue.

.

Stack VS Queue

Applications: Stacks are often used in applications where the order of operations is important, such as in undo/redo functionality in a text editor. Queues are often used in applications where elements are processed in the order they are received, such as in a message queue system.

Implementation: Both stacks and queues can be implemented using arrays or linked lists, but the implementation details may differ.



What is Heap

What is Heap

- Heap is a specialized tree-based data structure that is often used for implementing priority queues.
- A priority queue is a data structure that allows elements to be added with a priority value, and elements are removed from the queue in order of their priority values.

- Heaps are typically implemented as binary trees, where each node in the tree has a value and a priority.
- In a binary heap, the parent node has a higher priority value than its, children nodes. This ensures that the highest-priority element is always at the root of the heap.

What is Heap

- Python provides a built-in module called **‘heapq’** that includes functions for working with heaps.

- The ‘heapq’ module provides functions like ‘**heappush**’ to add elements to the heap, **‘heappop**’ to remove and return the highest-priority element from the heap, and ‘**heapify**’ to convert a list into a heap

What is Heap

Here's an example of how you can create and use a heap in Python using the heapq module:

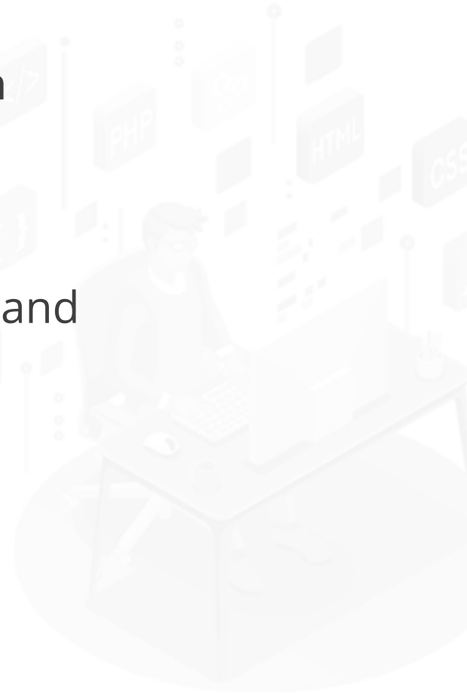
```
import heapq
create a new empty heap
        h = []
# add elements to the heap with priority values
        heapq.heappush(h, (3, "element 3"))
        heapq.heappush(h, (1, "element 1"))
        heapq.heappush(h, (2, "element 2"))
# remove elements from the heap in priority order
       first_element = heapq.heappop(h)
       second_element = heapq.heappop(h)
```

What is Map

What is Map

- In Python, **map()** is a built-in function that applies a given function to each item of an iterable (such as a list or a tuple) and returns a new iterable containing the results.

- The map() function takes two arguments: the first argument is the function to apply, and the second argument is iterable.



What is Map

Syntax for using the map() function:

map(function, iterable):

- The '**function**' argument is the function that will be applied to each item in the iterable. It can be any callable object such as a function, lambda function, or even a method of an object.
- The '**iterable**' argument is the sequence of items to which the function will be applied. This could be a list, tuple, set, or any other iterable object.
- The '**map()'** function returns an iterator, which can be converted into a list or tuple or consumed in a loop to get the result of applying the given function to each element of the iterable.

Byte Array

Byte Array

- In Python, a byte array is a mutable sequence of bytes. It is similar to a regular Python list or array, but instead of containing arbitrary objects, it contains a sequence of bytes.
- Byte arrays are useful when working with binary data, such as file I/O or network protocols.
- To create a byte array in Python, you can use the built-in '**bytearray()'** function.
- The 'bytearray()' function takes one argument, which can be an integer, a string, or an iterable of integers.

Byte Array

Here's an example of how to create a byte array using the **bytearray()** function:

```
# create a byte array from an integer
    a = bytearray(3)
    print(a)  # Output: bytearray(b'\x00\x00\x00')
# create a byte array from a string
     b = bytearray("hello", "utf-8")
     print(b)  # Output: bytearray(b'hello')
# create a byte array from an iterable of integers
    c = bytearray([1, 2, 3, 4, 5])
    print(c)  # Output: bytearray(b'\x01\x02\x03\x04\x05')
```

Binary Tree

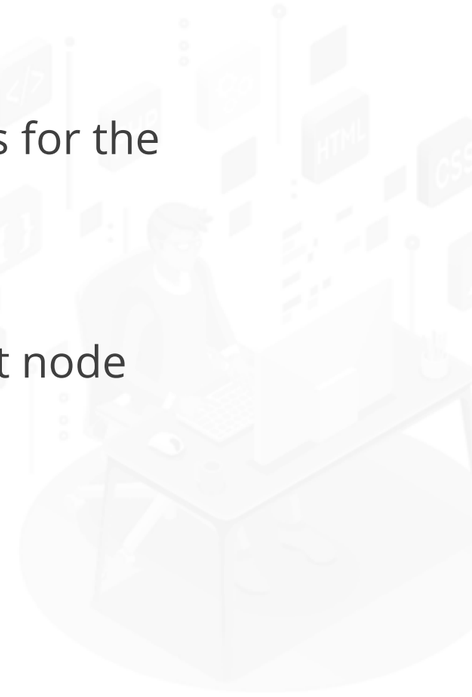
Binary Tree

- In Python, a binary tree is a data structure consisting of nodes, where each node has at most two child nodes (called the left child and the right child).
- Each node in a binary tree contains a value and a reference to its left and right child nodes.
- Binary trees are used to represent hierarchical data structures such as directories, file systems, and expressions in computer science.

Binary Tree

- To implement a binary tree in Python, you can create a class for the tree and a class for the nodes.

- Each node will have a value and references to its left and right child nodes. The root node of the tree will be an instance of the node class.



Key Takeaways

- Stack is a data structure that allows for last-in, first-out (LIFO) access.
- Heaps are typically implemented as binary trees, where each node in the tree has a value and a priority.
- In Python, a byte array is a mutable sequence of bytes.
- A binary tree is a data structure consisting of nodes, where each node has at most two child nodes.