In computer systems, especially those that use a segmented memory model, the terms "stack segment," "data segment," and "text segment" refer to distinct regions of a process's memory address space. Each segment serves a specific purpose and has its characteristics. These concepts are particularly relevant in older memory management models and architectures, such as the x86 architecture. Here's an overview of each segment:

1. **Text Segment (Code Segment)**:
   * **Purpose**: The text segment, often referred to as the code segment, contains the executable code of a program. It holds the machine code instructions that make up the program's logic.
   * **Characteristics**: This segment is typically marked as read-only because program code should not be modified during execution. It is also usually marked as executable, allowing the CPU to fetch and execute instructions from this segment.
   * **Examples**: The actual program instructions, functions, and procedures are stored in the text segment.
2. **Data Segment**:
   * **Purpose**: The data segment stores global and static variables, as well as initialized and uninitialized data used by a program. This includes both variables declared at the global scope and those defined within functions.
   * **Characteristics**: The data segment can be divided into further subsegments, such as the initialized data segment (initialized variables) and the uninitialized data segment (uninitialized variables). These segments can have different permissions (read-write) and characteristics.
   * **Examples**: Variables declared outside of functions, strings, arrays, and other data structures are typically stored in the data segment.
3. **Stack Segment (Stack)**:
   * **Purpose**: The stack segment is used for managing the program's runtime stack. It stores function call information, local variables, and return addresses for function calls.
   * **Characteristics**: The stack grows and shrinks dynamically as function calls are made and return. It is typically organized as a last-in-first-out (LIFO) data structure, and it allows for efficient function call and return operations.
   * **Examples**: Local variables, function call frames, and parameters are managed on the stack. The stack is crucial for maintaining program execution flow.