1) Heap ve Steak alanları nedir? Nasıl çalışır?

Stack bellek, **LIFO (Last In, First Out)** prensibiyle çalışan, hızlı ve düzenli bir bellek alanıdır. Yani son eklenen veri, ilk olarak silinir.. Bu bölgede **metod çağrıları, fonksiyon parametreleri ve yerel değişkenler** saklanır Bu yapı, özellikle fonksiyonların çalışma sürecini yönetmek için idealdir. Bir fonksiyon çağrıldığında, o fonksiyon için bir bellek bloğu ayrılır ve fonksiyon tamamlandığında bu blok otomatik olarak temizlenir. Bellek tahsisi ve temizliği derleyici tarafından kontrol edilir. Stack'in en önemli avantajı **hızıdır**. Ancak stack'in bir dezavantajı, **boyutunun sınırlı** olmasıdır. Eğer stack alanı aşılacak kadar çok veri eklenirse, **stack taşması (stack overflow)** hatası oluşur ve program çöker.

Heap bellek ise **dinamik bellek tahsisi** için kullanılan alandır. Stack'in aksine, **heap belleğe ayrılan veriler program sonlanana kadar bellekte kalabilir**, çünkü yönetimi tamamen programcıya bırakılmıştır. Java gibi dillerde çöp toplayıcı (**Garbage Collector**) sayesinde kullanılmayan heap belleği otomatik olarak temizlenir. Eğer programcı heap'te tahsis edilen belleği serbest bırakmazsa, **memory leak (bellek sızıntısı)** meydana gelebilir. Heap bellek, **LILO (Least In, Last Out)** veya **serbest bellek yönetimi** mantığıyla çalışır; yani bir veri belleğe erken yerleştirilmiş olsa bile, ne zaman temizleneceği tamamen programın akışına bağlıdır. Heap'in en kritik avantajı **büyük bellek alanlarına erişim** sağlamasıdır. Ancak bu esneklik, bazı riskleri de beraberinde getirir. Manuel yönetim nedeniyle **bellek sızıntıları (memory leak)** oluşabilir. Örneğin, heap'ten ayrılan bir bellek bloğu serbest bırakılmazsa, program çalıştığı sürece bu alan işgal edilir ve zamanla bellek tükenir.

- **Stack**, kısa ömürlü ve küçük boyutlu veriler için tercih edilir. Örneğin, bir döngü içinde kullanılan geçici değişkenler veya fonksiyon parametreleri.
- **Heap** ise uzun ömürlü veya büyük veriler için kullanılır. Örneğin, bir dosyadan okunan binlerce satırlık veri veya birden fazla fonksiyon tarafından erişilmesi gereken global bir nesne.

Kısacası, Stack, hızlı ve otomatik bellek yönetimi sunan ancak sınırlı boyutlu bir alanken; heap, esnek ve büyük ölçekli veriler için tasarlan ancak dikkatli yönetilmesi gereken bir bellek bölgesidir. Doğru tercih, programın performansı ve güvenliği açısından kritik öneme sahiptir.

2) Garbage collector nedir? Ne işe yarar? Nasıl çalışır?

Garbage Collector (Çöp Toplayıcı), programlama dillerinde bellek yönetimini otomatik hale getiren bir sistemdir. Temel amacı, programın çalışması sırasında artık kullanılmayan ve bellekte gereksiz yer kaplayan nesneleri tespit edip temizleyerek bellek sızıntılarını önlemektir. Bu sayede geliştiricilerin manuel olarak bellek ayırma ve serbest bırakma işlemleriyle uğraşmasına gerek kalmaz, bu da kodun daha güvenli ve hatasız olmasını sağlar. Çalışma mantığı genellikle iki temel adıma dayanır: İşaretleme (Mark) ve Temizleme (Sweep). İlk aşamada, programın aktif olarak kullandığı referanslardan (örneğin global değişkenler veya stack'teki yerel değişkenler) yola çıkarak tüm erişilebilir nesneler işaretlenir. İkinci aşamada ise işaretlenmemiş olan (yani hiçbir referans tarafından gösterilmeyen) nesneler bellekten silinir ve bu alan yeniden kullanılmak üzere sisteme geri verilir. Java, C# ve Python gibi modern programlama dillerinde Garbage Collector otomatik olarak çalışırken, C ve C++ gibi dillerde bellek yönetimi programcı tarafından manuel olarak yapılır. Ancak Garbage Collector'ın bir dezavantajı, çalıştığı sırada programın geçici olarak duraklamasıdır. Özellikle gerçek zamanlı sistemlerde veya yüksek performans gerektiren uygulamalarda bu duraklamalar sorun yaratabilir.

Kısacası, Garbage Collector bellek yönetimini otomatikleştirerek hem zaman kazandırır hem de kritik hataları önler. Ancak performans üzerindeki etkisi nedeniyle dilin ve uygulamanın ihtiyaçlarına göre dengeli bir şekilde kullanılmalıdır.

3) x=2 tanımladığımda RAM'in neresinde tanımlanır? Liste tanımladığımda ne heap'te, ne stackte yer alır?

Programlama dillerinde değişken atamaları ve veri yapıları bellekte farklı şekillerde yönetilir. Örneğin, Python'da x = 2 dediğinizde, x isminde bir yerel değişken, referans oluşturulur. Bu referans, değeri 2 olan nesneye işaret eder. Python'da nesneler heap bellekte saklanır. Ancak, x isminden bahsettiğimizde, bu isim yerel isimler sözlüğünde tutulur ki bu yapı da genellikle fonksiyon çağrıları sırasında stack üzerinde yönetilir. Benzer şekilde, arabalar = ("ford", "dodge") ifadesiyle bir tuple tanımladığınızda, tuple nesnesi dinamik olarak oluşturulur ve heap bellekte yer alır. Bu nesneye işaret eden arabalar isimli referans ise yerel değişkenler arasında bulunur ve yine stack belleğin bir parçası olarak yönetilir.

Özetle, Python'da nesnelerin kendileri (2 veya tuple) heap bellekte depolanırken, bu nesnelere referans veren yerel değişkenler (x veya arabalar) stack üzerinde tutulur.

Değişken/Değer	Stack'te mi?	Heap'te mi?
`x`	✔ (adres)	✘
`2`	✘	✔
`arabalar`	✔ (adres)	✘
`("ford", "dodge")`	✘	✔
`"ford"`	✘	✔