# .NET/C# Syntax.

**Modern .NET/C# Course**

**.NET Academy**

# Content

- Value Types and Reference Types.
- Nullable types.
- Null operators.
- ● Where does variables are allocated?

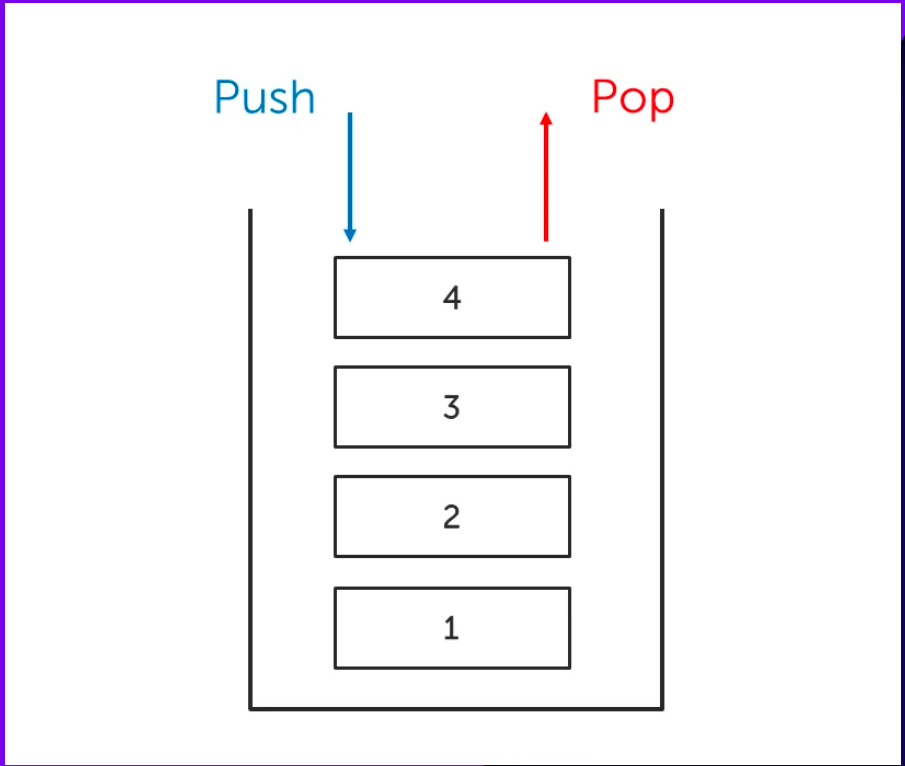
# Value Types and Reference Types

So, when we have been talked about variables and data types, we need to make clear where does they stores, and how does memory is managed in C#.

"The stack" refers to the call stack, which is an implementation of the stack data structure. There are two different levels to explaining the stack: first there is the generic stack data structure, then there is the call stack, which is intrinsic to the C# runtime. Let's first understand the stack data structure and then see how the call stack implements the data structure for performing its function in memory management.

The defining property of stacks is that they're last-in, first-out, meaning the last item added to the stack is the first item to be removed. In other words, to place a new item onto a stack it has to go on the top, and only the item currently at the top can be removed, therefore if you wanted to remove an item from the middle of a stack, say, the third item down from the top, you would first have to remove the top two items.

# Value Types and Reference Types




.NET Academy

# Value Types and Reference Types

A stack of plates is a good analogy for understanding the stack data structure – to get a plate you must take one from the top (pop), to add a plate you must add it to the top of the stack (push). The stack is a simple data structure and its two operations – push and pop – are fast and efficient.
Again, the call stack is an implementation of a stack and is an intrinsic part of the C# runtime; it is an important component in the memory management of C# programs.
The stack has two main purposes:
(1) to keep track of the method that control should return to once the currently executing method has finished
(2) to hold the values of local variables (i.e. the variables that are not needed once their containing method finishes executing).

# Value Types and Reference Types

As local variables are declared inside a method they're pushed onto the stack (just like I described for the generic stack data structure). The local variables pertaining to a given method are grouped together in what's called a stack frame; when a method finishes executing, the corresponding stack frame is popped from the stack, meaning all of the variables contained within are removed together and become unavailable. This results in the stack frame from the previous method being at the top of the stack, and, consequently, the local variables it contains being in scope – this is how variable scope is managed in C#. The examples and their associated diagrams shown later will help to understand this process.

Now, let’s talk about second memory type - Heap. Heap refers to the managed heap, which, like the stack, plays an important (but different) role in the memory management of C# programs. The heap is an intrinsic part of the C# runtime and is an implementation of a heap data structure.
Whereas the stack only allows items to be added and removed to/from the top, any item in a heap can be accessed at any time. This additional flexibility of the heap compared to the stack makes it a more complex data structure with higher overheads for managing the items that it holds.

# Value Types and Reference Types

The purpose of the heap is to store data that needs to outlive specific methods. This means the heap is used to store reference type variables, which are referred to as objects. The garbage collector (GC) is a program that manages the objects on the heap, it allocates objects and reclaims objects that are no longer being used - freeing memory for future allocations.

Each C# variable has a specific type, and the contents of the memory block associated with a given variable depend on its type. C# variables fall into two distinct type categories – value types and reference types, which are handled differently in memory. The type of a variable – specifically whether it's a reference or value type – and the context in which it was declared, determine whether it is stored on the stack or heap.

# Where does variables are allocated?

C# variables are stored on either the stack or heap, which one depends on whether the variable is of reference or value type, and on the context in which the variable is declared.

- Local variables (i.e. those that are declared inside methods) are stored on the stack. This means their values are stored on the stack, therefore meaning that local reference type variables have references stored on the stack and local value type variables have actual values stored on the stack.
- Objects of reference type variables (i.e. the things that references point to) always live on the heap.
- Instance variables that are part of a reference type instance (e.g. a field on a class) are stored on the heap with the object itself.
- Instance variables that are part of a value type instance are stored in the same context as the variable that declares the value type. This means that a variable of a struct that is declared in a method will live on the stack, whilst a variable of a struct that is declared inside a class (i.e. a field on the class) will live on the heap.

# Where does variables are allocated?

Static variables always live on a heap, and there's only ever a single heap block that holds the value of the variable. This makes sense when you think about what static variables are: there's a single object that the variable refers to which needs to outlive the method in which it was declared.

# Null Types

A nullable value type T? represents all values of its underlying value type T and an additional null value. For example, you can assign any of the following three values to a bool? variable: true, false, or null. An underlying value type T cannot be a nullable value type itself.
Any nullable value type is an instance of the generic System.Nullable<T> structure. You can refer to a nullable value type with an underlying type T in any of the following interchangeable forms: Nullable<T> or T?.

.NET Academy

# Null Types

We typically use a nullable value type when we need to represent the undefined value of an underlying value type. For example, a Boolean, or bool, variable can only be either true or false. However, in some applications a variable value can be undefined or missing. For example, a database field may contain true or false, or it may contain no value at all, that is, NULL. We can use the bool? type in that scenario.

# Null operators

The null-coalescing operator ?? returns the value of its left-hand operand if it isn't null; otherwise, it evaluates the right-hand operand and returns its result. The ?? operator doesn't evaluate its right-hand operand if the left-hand operand evaluates to non-null. The null-coalescing assignment operator ??= assigns the value of its right-hand operand to its left-hand operand only if the left-hand operand evaluates to null. The ??= operator doesn't evaluate its right-hand operand if the left-hand operand evaluates to non-null.

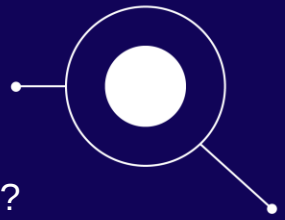
```
List<int> numbers = null;
int? a = null;

Console.WriteLine((numbers is null)); // expected: true
// if numbers is null, initialize it. Then, add 5 to numbers
(numbers ??= new List<int>()).Add(5);
Console.WriteLine(string.Join(" ", numbers));  // output: 5
Console.WriteLine((numbers is null)); // expected: false


Console.WriteLine((a is null)); // expected: true
Console.WriteLine((a ?? 3)); // expected: 3 since a is still null
// if a is null then assign 0 to a and add a to the list
numbers.Add(a ??= 0);
Console.WriteLine((a is null)); // expected: false
Console.WriteLine(string.Join(" ", numbers));  // output: 5 0
Console.WriteLine(a);  // output: 0
```

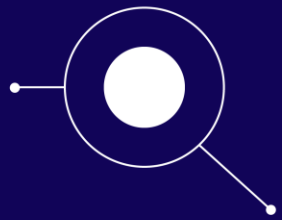
.NET Academy

# Questions for the study of the topic



1. What is the main difference between Value Types and Reference Types in C#?
2. What examples of Value Types do you know? Give some examples.
3. What examples of Reference Types can you name? Give some examples.
4. What are Nullable Types, and why are they used?
5. How to declare a variable with Nullable type in C#?
6. How does the '?'. operator (null-conditional operator) work in C#? What tasks does it solve?
7. How is the '??' operator (null-coalescing operator) used? What is it used for?
8. What are HEAP and STACK memory in the context of .NET?
9. What data is typically stored in HEAP memory, and how is it managed?

# Homework



1. Why do Value Types store data directly and Reference Types store references?
2. Which data types are usually Value Types and which are Reference Types in C#?
3. How to declare a variable with Nullable type? What is its default value?
4. How can you check if a Nullable variable has a value of null?
5. In what situations can the ??= operator (null-reference operator) be useful? Give examples.
6. What does the ??? (null-coalescing operator)? How does it help to handle null values?
7. What data is typically stored in HEAP memory, and how is it managed in .NET?
8. What data is stored in STACK memory, and how does it differ from data in HEAP?
9. Where are the Value Types and Reference Types variables allocated in .NET? Which variables are allocated in STACK and which are allocated in HEAP?

Thank you for your attention.

.NET Academy