Τι είναι η C# – Αναλυτική Εισαγωγή

Η C# είναι μια σύγχρονη αντικειμενοστρεφής γλώσσα προγραμματισμού που δημιουργήθηκε από τη Microsoft.
Ανήκει στην πλατφόρμα .NET και χρησιμοποιείται για ανάπτυξη:

- Desktop εφαρμογών
- Web εφαρμογών
- APIs
- Game development (Unity)
- Mobile εφαρμογών (μέσω .NET MAUI)

Το βασικό χαρακτηριστικό της C# είναι ότι είναι strongly typed γλώσσα.
Αυτό σημαίνει ότι κάθε μεταβλητή έχει συγκεκριμένο τύπο δεδομένων. Δεν μπορούμε να αλλάζουμε τύπο αυθαίρετα.

Η C# δεν εκτελείται απευθείας από το λειτουργικό σύστημα.
Μεταγλωττίζεται σε ενδιάμεσο κώδικα (IL – Intermediate Language) και εκτελείται από το .NET runtime.

Αυτό προσφέρει:

- Έλεγχο μνήμης
- Διαχείριση απορριμμάτων (Garbage Collection)
- Ασφαλέστερη εκτέλεση

◆ Η Δομή ενός Προγράμματος

Κάθε πρόγραμμα C# ξεκινά από την μέθοδο Main.

```
using System;

class Program
{
    static void Main()
    {
        Console.WriteLine("Καλημέρα");
    }
}
```

Τι γίνεται εδώ;

- `using System;` εισάγει βιβλιοθήκη.
- `class Program` δηλώνει μια κλάση.
- `static void Main()` είναι το σημείο εκκίνησης.
- `Console.WriteLine()` εμφανίζει μήνυμα.

Η C# είναι οργανωμένη σε κλάσεις. Δεν υπάρχουν “ελεύθερες” συναρτήσεις όπως σε C.

◆ Μεταβλητές και Τύποι

Η μεταβλητή είναι δεσμευμένος χώρος στη μνήμη που αποθηκεύει τιμή.

Γενική μορφή:

```
τύπος όνομα = τιμή;
```

Παραδείγματα:

```
int age = 25;
double price = 19.99;
string name = "Νίκος";
bool isPassed = true;
```

Εδώ διδάσκεις:

- int = ακέραιος
- double = δεκαδικός
- string = κείμενο
- bool = λογική τιμή

Η C# απαιτεί να δηλώσεις τύπο.

Αυτό αποτρέπει πολλά λάθη.

◆ Value Types και Reference Types

Εδώ κάνεις σοβαρή διάκριση.

Value Types

Αποθηκεύουν κατευθείαν την τιμή.

Όταν αντιγράφονται, δημιουργείται νέο αντίγραφο.

```
int a = 5;
int b = a;
b = 10;
```

Το a παραμένει 5.

Reference Types

Αποθηκεύουν αναφορά (δείκτη) στη μνήμη.

```
Student s1 = new Student();
Student s2 = s1;
```

Τώρα δείχνουν στο ίδιο αντικείμενο.

Αλλαγή μέσω του ενός επηρεάζει και το άλλο.

Αυτό είναι θεμελιώδες για το OOP.

◆ Δομές Επιλογής

Οι συνθήκες ελέγχουν τη ροή.

```
if (grade >= 10)
{
    Console.WriteLine("Πέρασε");
}
else
{
    Console.WriteLine("Κόπηκε");
}
```

Εδώ εισάγεις:

- Σύγκριση
- Boolean αποτέλεσμα
- Διακλαδώσεις

◆ Επαναλήψεις

Η επανάληψη επιτρέπει εκτέλεση κώδικα πολλές φορές.

```
for (int i = 0; i < 5; i++)
{
    Console.WriteLine(i);
}
```

Εξηγείς:

- Αρχικοποίηση

- Συνθήκη
- Αύξηση

Η while εκτελείται όσο ισχύει συνθήκη.

◆ Πίνακες

Ο πίνακας είναι σταθερό σύνολο στοιχείων ίδιου τύπου.

```
int[] numbers = { 10, 20, 30 };
```

Έχουν μήκος και δεν αλλάζει.

Για δυναμική συλλογή χρησιμοποιούμε List.

◆ Μέθοδοι

Οι μέθοδοι οργανώνουν τη λογική.

```
static int Add(int a, int b)
{
    return a + b;
}
```

Τονίζεις:

- Επιστρέφει τιμή
- Έχει παραμέτρους
- Βελτιώνει την επαναχρησιμοποίηση

◆ Κλάσεις

Η κλάση είναι πρότυπο.

```
class Student
{
    public string Name { get; set; }
    public int Grade { get; set; }
}
```

Το αντικείμενο είναι πραγματικό στιγμιότυπο.

```
Student s = new Student();
```

Εδώ μπαίνει η έννοια του OOP.

◆ Ενθυλάκωση

Δεν εκθέτουμε δεδομένα ανεξέλεγκτα.

```
private int grade;

public int Grade
{
    get { return grade; }
    set
    {
        if (value >= 0 && value <= 20)
            grade = value;
    }
}
```

Εδώ ελέγχεις τα δεδομένα πριν μπουν στο αντικείμενο.

◆ Κληρονομικότητα

Η μία κλάση μπορεί να επεκτείνει άλλη.

```
class Person { }
class Student : Person { }
```

Η Student κληρονομεί ιδιότητες της Person.

◆ Πολυμορφισμός

Διαφορετική συμπεριφορά με ίδια μέθοδο.

```
public virtual void Speak() { }
public override void Speak() { }
```

Σημαντικό για αρχιτεκτονική.

◆ Εξαιρέσεις

Προστασία από λάθη εκτέλεσης.

```
try
{
    int x = int.Parse(input);
}
catch (Exception)
{
    Console.WriteLine("Σφάλμα");
}
```

Δεν αφήνουμε το πρόγραμμα να καταρρεύσει.

◆ LINQ

Σύγχρονος τρόπος επεξεργασίας συλλογών.

```
var evens = numbers.Where(n => n % 2 == 0);
```

Γίνεται πιο εκφραστική η γλώσσα.