1. Eljárásorientált programozási nyelvek

Az eljárásorientált nyelvek a programok eljárásokra, azaz funkciókra, eljárásokra bontására építenek. Az eljárások egymás után, sorban végrehajtva oldják meg a problémákat. A programozók a változókat és adatokat külön kezelik, míg az eljárások a műveleteket hajtják végre. Az eljárásorientált megközelítésben a kód strukturált és könnyen érthető. **Példa nyelv**: C, Pascal

2. Objektumorientált programozási nyelvek

Az objektumorientált programozás (OOP) alapelve az objektumok létrehozása, amelyek tartalmazzák az adatok (mezők) és a hozzájuk kapcsolódó műveletek (metódusok) kombinációját. Az OOP a reusability (újrahasználhatóság), az öröklés és a polimorfizmus alapelveire épít. Az OOP előnye, hogy a kód jobban strukturálható és karbantartható.

```
class Car {
    public string brand; // Az autó márkája
    public void Start() {
        Console.WriteLine("The car starts."); // Az autó indítása
    }
}
```

3. Szekvencia

A szekvencia az utasítások végrehajtásának sorrendje. A programok általában egy szekvenciával kezdődnek, amely lehetővé teszi a változók inicializálását és más utasítások végrehajtását egymás után. A szekvenciális végrehajtás biztosítja, hogy az utasítások a kívánt sorrendben történjenek.

```
int a = 10; // Változó inicializálása
int b = 20; // Még egy változó
int c = a + b; // Összeadás végrehajtása
Console.WriteLine(c); // Eredmény kiírása
```

4. Szintaxis

A szintaxis a programozási nyelv szabályait jelenti, amelyek meghatározzák, hogy hogyan kell a kódot helyesen megírni. Minden programozási nyelvnek megvan a saját szintaxisa, és a hibás szintaxis hibát okoz a fordítás során. A szintaxis magában foglalja a kulcsszavak, az azonosítók és az operátorok helyes használatát.

5. Névterek

A névterek segítenek elválasztani az azonos nevű osztályokat, változókat és egyéb elemeket, hogy elkerüljük az ütközéseket. A névterek használata lehetővé teszi a kód moduláris felépítését, ami megkönnyíti a nagyméretű projektek kezelését. A névtér elérhetősége változó lehet, ezért a használatukkor figyelembe kell venni a láthatóságukat is.

```
using System;  // A System névtér használata
```

6. Kulcsszavak

A kulcsszavak előre definiált szavak, amelyeknek speciális jelentése van a programozási nyelvben. Ezek nem használhatók változók vagy azonosítók neveként. A kulcsszavak segítik a nyelv szintaxisának meghatározását, és az OOP elveit is tükrözik. **Példa**: int, class, public, void.

7. Azonosító

Az azonosítók a programban használt változók, függvények, osztályok és egyéb elemek nevei. Az azonosítók használata lehetővé teszi a programozók számára, hogy hivatkozzanak különböző elemekre a kódban. Az azonosítóknak egyedieknek kell lenniük a saját hatáskörükben.
Példa: myVariable, CalculateTotal().

8. Változók

A változók olyan adattárolók, amelyek segítségével adatokat tudunk tárolni és kezelni. A változók típusai határozzák meg, hogy milyen típusú adatokat tárolhatunk bennük (pl. számok, karakterek, logikai értékek). A változók neve, típusa és értéke van. **Példa**: int age = 25; **Komponensek**: név (pl. age), típus (pl. int), érték (pl. 25).

9. Operátorok

Az operátorok olyan szimbólumok, amelyek műveleteket hajtanak végre operandusokon. Az operátorok lehetnek aritmetikai (pl. +, -), logikai (pl. &&, ||) vagy relációs (pl. ==, <). Az operátorok lehetővé teszik a matematikai műveletek végrehajtását, valamint a logikai kifejezések kiértékelését.

```
int sum = 10 + 5;  // Aritmetikai operátor: összegzés
bool isEqual = (a == b); // Relációs operátor: egyenlőség vizsgálata
```

10. Blokk

A blokk egy olyan kódrészlet, amelyet {} jelek határolnak. A blokkban több utasítást is végrehajthatunk. A blokkok használata lehetővé teszi a kód logikai csoportosítását, és segít a program struktúrájának megőrzésében.

```
{
    int a = 10; // Változó inicializálása a blokkban
    int b = 20; // Másik változó
    Console.WriteLine(a + b); // Kód a blokkban
}
```

11. Alprogram

Az alprogram olyan programrészlet, amely egy adott feladatot hajt végre, és lehetőséget ad a kód újrafelhasználására. Az alprogramok segítenek a kód modulárisabbá tételében és a karbantartás megkönnyítésében.

12. Metódus

A metódusok olyan eljárások, amelyek az osztályokon belül találhatók, és amelyek meghatározzák az osztály által végrehajtható műveleteket. A metódusoknak lehetnek paramétereik, és visszaadhatnak értéket, vagy eljárásként működhetnek.

```
public void DisplayMessage() {
    Console.WriteLine("Hello!"); // Üzenet megjelenítése
}
```

13. Függvény

A függvény olyan metódus, amely visszaad egy értéket. A függvények általában egy bemeneti értéket fogadnak el, és visszaadnak egy számított vagy manipulált értéket.

```
public int Add(int a, int b) {
    return a + b; // Összegzés és visszaadás
}
```

14. Eljárás

Az eljárás olyan metódus, amely nem ad vissza értéket, csak végrehajt bizonyos utasításokat. Az eljárások gyakran használatosak mellékhatások (pl. üzenetek megjelenítése) előállítására.

```
public void Greet() {
    Console.WriteLine("Hello!"); // Üdvözlő üzenet
}
```

15. Mező

A mezők az osztály adatait tároló változók. A mezők tartalmazzák az osztály állapotát, és más osztályokból is elérhetők, ha megfelelően kezeljük a láthatóságukat.

```
class Car {
    public string brand; // Az autó márkája
}
```

16. Property

A property egy speciális mező, amely getter és setter metódusokkal rendelkezik, lehetővé téve az adatok védelmét és az érvényesítést. A property-k segítségével kontrollálhatjuk, hogy az osztályon belül hogyan érhetjük el az adatokat.

```
private string _brand;
public string Brand {
    get { return _brand; }
    set { _brand = value; }
}
```

17. Getter

A getter metódus egy property-hez tartozó olvasó metódus, amely lehetővé teszi az adatok elérését anélkül, hogy közvetlenül manipulálnánk a mezőt.

```
public string Brand {
    get { return _brand; }
}
```

18. Setter

A setter metódus egy property-hez tartozó író metódus, amely lehetővé teszi a mező értékének módosítását, és lehetőséget ad az érvényesítésre is.

```
public string Brand {
    set { _brand = value; } // Érték beállítása
}
```

19. Elágazások

Az elágazások lehetővé teszik, hogy a program különböző útvonalakat kövessen a logikai feltételek függvényében. Az elágazások lehetnek kétirányúak (if-else) vagy több irányúak (switch-case).

```
if (age >= 18) {
    Console.WriteLine("Adult");
} else {
    Console.WriteLine("Minor");
}
```

20. Integrált fejlesztői környezet (IDE)

Az IDE egy szoftvereszköz, amely segíti a programozókat a kód írásában, tesztelésében és hibakeresésében. Az IDE-k általában tartalmaznak kódszerkesztőt, fordítót, debugger-t és egyéb eszközöket, amelyek megkönnyítik a programozást. **Példa**: Visual Studio, JetBrains Rider.

21. Ciklusok

A ciklusok lehetővé teszik, hogy egy adott kódrészletet többször is végrehajtsunk, amíg egy bizonyos feltétel teljesül. A ciklusok típusai közé tartozik a for, while és do-while ciklus.

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

22. Literál

A literál egy konkrét, közvetlenül a kódban megadott érték. A literálok például számok, karakterek, szövegek, stb.

```
int num = 42; // Egész szám literál
string text = "Hello"; // Szöveg literál
```

23. Megjegyzés

A megjegyzések olyan szövegrészek, amelyeket a fordító figyelmen kívül hagy. A megjegyzések segítenek a kód dokumentálásában, és megkönnyítik a programozók számára a kód megértését.

```
// Ez egy egysoros megjegyzés
/*
Ez egy
többsoros
megjegyzés
*/
```

24. Adattípusok

Az adattípusok határozzák meg, hogy egy változó milyen típusú adatot tárolhat. Az adattípusok csoportosíthatók egyszerű (pl. int, char) és összetett típusokra (pl. osztályok, struktúrák).

```
int number = 10; // Egyszerű adattípus
List<int> numbers = new List<int>(); // Összetett adattípus
```

25. Paraméterátadás

A paraméterátadás a függvényekhez és metódusokhoz kapcsolódó értékek átadását jelenti. A paramétereket általában a függvény vagy metódus definíciójában határozzák meg, és a hívás során átadják őket.

```
public void PrintMessage(string message) {
    Console.WriteLine(message); // Paraméter átadása
}
```

26. Kifejezések

A kifejezések a programozásban olyan kombinációk, amelyek operátorokból és operandusokból állnak, és egy értéket eredményeznek. A kifejezések lehetnek aritmetikai, logikai vagy relációs kifejezések.

```
int result = 10 + 5 * 2; // Aritmetikai kifejezés
```

27. Hatáskör

A hatáskör a változók és metódusok érvényességi területe. A lokális változók csak a blokkon belül érhetők el, míg a globális változók az egész programban elérhetők.

```
void Method() {
    int localVar = 10; // Lokális változó
}
// globalVar is not accessible here
```

28. Kivételkezelés

A kivételkezelés lehetővé teszi a program számára, hogy kezelje a váratlan helyzeteket és hibákat. A try, catch és finally blokkok segítenek a hibák kezelésében, és lehetővé teszik a program stabil működését.

```
try {
    int result = 10 / 0; // Nullával való osztás
} catch (DivideByZeroException e) {
    Console.WriteLine("Error: " + e.Message);
}
```

29. Tiszta kód

A tiszta kód az olvashatóságra és karbantartásra összpontosít. A tiszta kód alapelvei közé tartozik a megfelelő elnevezés, a kód rövidsége, a megjegyzések használata és a kód modularizálása. A tiszta kód segít a csapatmunkában és a hosszú távú projekteknél.

30. Osztály

Az osztály egy absztrakt adatstruktúra, amely a hasonló objektumok közös jellemzőit és működéseit tartalmazza. Az osztályok mezőkből és metódusokból állnak, amelyek meghatározzák az osztály állapotát és viselkedését.

```
class Person {
    public string Name; // Mező
    public void Greet() {
        Console.WriteLine("Hello, " + Name); // Metódus
    }
}
```

31. Öröklődés

Az öröklődés lehetővé teszi egy osztály számára, hogy egy másik osztály jellemzőit és működéseit örökölje. Ezáltal csökkenthető a kód ismétlése és lehetővé válik a közös viselkedés definiálása.

```
class Animal {
    public void Eat() { Console.WriteLine("Eating..."); }
}
class Dog : Animal { // Öröklés
    public void Bark() { Console.WriteLine("Barking..."); }
}
```

32. Polimorfizmus

A polimorfizmus lehetővé teszi, hogy ugyanaz a metódus különböző osztályokban különböző módon viselkedjen. A polimorfizmus két fő típusa a futásidejű polimorfizmus (metódusok felülírása) és a fordítási idejű polimorfizmus (metódusok túlterhelése).

```
class Animal {
    public virtual void Sound() { Console.WriteLine("Animal sound"); }
}
class Dog : Animal {
    public override void Sound() { Console.WriteLine("Bark"); } // Polimorfizmus
}
```

33. Konstruktor

A konstruktor egy különleges metódus, amelyet az osztály példányosításakor hívnak meg. A konstruktorok célja az objektumok kezdeti állapotának beállítása.

```
class Car {
    public string Model;
    public Car(string model) { // Konstruktor
        Model = model; // Mező inicializálása
    }
}
```

34. Lokális változó

A lokális változó olyan változó, amelyet egy blokkon belül definiálnak, és amely csak azon a területen érhető el, ahol létrejött. A lokális változók a blokk végrehajtásakor keletkeznek, és a blokk végén megszűnnek.

```
void MyMethod() {
    int localVar = 5; // Lokális változó
    Console.WriteLine(localVar); // Lokális változó elérése
}
```

35. Globális változó

A globális változó olyan változó, amely az egész program területén elérhető. A globális változók általában a program elején, a fő metóduson kívül vannak definiálva. A globális változók használata gyakran nem ajánlott, mivel nehezen nyomon követhetők és hibákhoz vezethetnek.

```
int globalVar = 10; // Globális változó
void MyMethod() {
    Console.WriteLine(globalVar); // Globális változó elérése
}
```

36. Objektum

Az objektum az osztály példánya, amely tartalmazza az osztály mezőit és metódusait. Az objektumok valós világbeli entitásokat képviselnek, és képesek interakcióba lépni más objektumokkal.

```
Car myCar = new Car("Tesla"); // Objektum létrehozása
myCar.Start(); // Metódus hívása
```

37. Nevesített konstans

A nevesített konstans olyan változó, amelynek értéke állandó, és amelyet a program futása során nem lehet módosítani. A nevesített konstansok segítenek az értékek könnyebb azonosításában.

```
const double Pi = 3.14; // Nevesített konstans
```

38. Utasítás

Az utasítások a programozási nyelvben végrehajtandó műveletek. Az utasítások végrehajtása általában a program logikai folyamatait határozza meg, például változók inicializálását, ciklusok és elágazások alkalmazását.

```
Console.WriteLine("Hello, World!"); // Utasítás végrehajtása
```

39. Interfész

Az interfész egy absztrakt típus, amely a hozzá tartozó metódusok aláírásait definiálja, de nem tartalmazhat megvalósítást. Az interfészek lehetővé teszik a különböző osztályok közötti kapcsolatok definiálását, és segítenek a kód újrafelhasználásában.

```
interface IDriveable {
    void Drive(); // Interfész metódus
}
class Car : IDriveable {
    public void Drive() {
        Console.WriteLine("Car is driving.");
    }
}
```