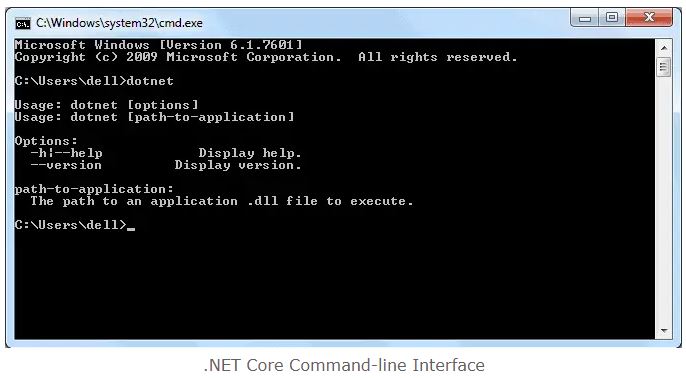
**C# .Net Core**

**.NET Core Command-Line Interface**

**یک ابزار cross-platform جدید است که برای ایجاد (creating)، بازیابی بسته‌ها (restoring packages)، ساختن (building)، اجرا (running) و انتشار (publishing) اپلیکیشن‌های دات نت به کار می‌رود.**

**این CLI همراه با نصب .Net Core SDK نصب می‌شود، بنابراین نیازی به نصب جداگانه ندارد. برای بررسی اینکه آیا این CLI در سیستم ما نصب شده است یا خیر، به این روش بررسی می‌کنیم. وارد command prompt یا همان cmd می‌شویم و تایپ می‌کنیم dotnet سپس اینتر را می‌زنیم.**

****

**ساختار اجرای آن به این صورت است که تمام دستورات آن ابتدا با dotnet شروع می‌شوند. یعنی برای اجرای یک دستور اول تایپ می‌کنیم dotnet و در ادامه اش دستور مورد نظر را وارد می‌کنیم. برای برای اجرای یک کنسول اپلیکیش به این صورت دستور را وارد می‌کنیم:** dotnet new console

**و اگر بخواهیم برای کنسول اپلیکیشن خودمان اسمی قرار دهیم به این صورت وارد می کنیم:**

dotnet new console -n MyConsoleApp

**سوئیج -n به معنای نام (name) می‌باشد.**

**جهت اجرای پروژه دستور dotnet run و یا dotnet watch run را اجرا می‌کنیم.**

**dotnet restore: وقتی پکیج جدید نصب می‌کنیم بهتر است که این دستور را اجرا کنیم.**

**اگر می‌خواهید که کنسول اپلیکیشن خود را با استفاده از dot net 6 اما با فرمت قدیمی استفاده کنید، دستور زیر را به کار می‌بریم:** dotnet new console --use-program-main

**فرمت قدیمی:**

using System;

namespace Helloworld

{

class Program

{

static void Main(string[] args)

{

Console.WriteLine("Hello World!");

}

}

}

**تعریف متغیر**

**Syntax (سینتکس) تعریف متغیر:**

<data type> <variable name> = <value>;

Ex: int num = 100;

**در مثال بالا،** int **نشان دهنده data type است. num نشان دهند نام متغیر که به آن identifier هم میگویند. علامت = جهت تخصیص دادن مقدار و عدد 100 هم مقدار یا همان value**

**قوانین نام گذاری متغییرها:**

* **نام متغیر باید با حروف یا undersore آغاز شود**
* **نام متغیر case sensitive است یعنی به حرف کوچک وبزرگ حساس است**
* **استفاده از keywordها در نام گذاری متغیرها مجاز نیست**

**نکته: در سی شارپ می‌توان ابتدا متغیر را تعریف کرد و سپس به آن مقدار دهی کرد. مثال زیر:**

int num;

num = 50;

**نکته: بعد از تعریف یک متغیر می‌توان مقدار آن را تغییر داد. مثال زیر:**

int num = 100;

num = 200;

Console.WriteLine(num); //output: 200

**تعریف متغیر به شکل explicitly**

**در سی شارپ، متغیر باید با data type (نوع) مشخص شود. به این حالت می‌گویند explicitly typed. یعنی نوع متغیر صراحتا مشخصا شده است. مثلا:**

int i = 100;// explicitly typed variable

**تعریف متغیر به شکل implicitly**

**در ورژن 3 سی شارپ، کلمه کلیدی (keyword) به نام var معرفی شد، که اجازه تعریف متغیر بدون تعیین data type رو می‌دهد. مثال:**

var j = 100; // implicitly typed local variable

**معرفی Data Typeها (انواع متغیرها)**

|  |  |  |  |
| --- | --- | --- | --- |
| byte | 8-bit unsigned integer | 0 to 255 |  |
| sbyte | 8-bit signed integer | -128 to 127 |  |
| short | 16-bit signed integer | -32,768 to 32,767 |  |
| ushort | 16-bit unsigned integer | 0 to 65,535 |  |
| int | 32-bit signed integer | -2,147,483,648 to 2,147,483,647 |  |
| uint | 32-bit unsigned integer | 0 to 4,294,967,295 | u |
| long | 64-bit signed integer | -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807 | l |
| ulong | 64-bit unsigned integer | 0 to 18,446,744,073,709,551,615 | ul |
| float | 32-bit Single-precision floating point type | -3.402823e38 to 3.402823e38 | f |
| double | 64-bit double-precision floating point type | -1.79769313486232e308 to 1.79769313486232e308 | d |
| decimal | 128-bit decimal type for financial and monetary calculations | (+ or -)1.0 x 10e-28 to 7.9 x 10e28 | m |
| char | 16-bit single Unicode character | Any valid character, e.g. a,\*, \x0058 (hex), or\u0058 (Unicode) |  |
| bool | 8-bit logical true/false value | True or False |  |
| object | Base type of all other types. |  |  |
| string | A sequence of Unicode characters |  |  |
| DateTime | Represents date and time | 0:00:00am 1/1/01 to 11:59:59pm 12/31/9999 |  |

**مثال برای داده های عددی:**

uint ui = 100u;

float fl = 10.2f;

long l = 45755452222222l;

ulong ul = 45755452222222ul;

double d = 11452222.555d;

decimal mon = 1000.15m;

**نکته: هر data type دارای یک مقدار پیش فرض می‌باشد. مقادیر پیش فرض در زیر آمده است:**

int i = default(int); // 0

float f = default(float);// 0

decimal d = default(decimal);// 0

bool b = default(bool);// false

char c = default(char);// '\0'

**تبدیل صریح – explicit conversion**

int i = 100;

uint u = (uint) i;

Console.Write(i);

**در مثال بالا متغیر i را که از نوع int است را به صورت explicit به نوع uint تبدیل کردیم.**

**اعداد در سی شارپ – number in c#**

**در سی شارپ اعداد به دو قسمت تقسیم می‌شوند. هinteger و floating-point (اعداد صحیح و اعداد اعشاری)**

**integer: اعداد صحیح را می‌گویند. عدد صحیح عددی است که بدون نماد کسری نوشته شده باشد. در واقع عدد صحیح قسمت اعشار ندارد. اعداد صحیح شامل عدد صفر، اعداد مثب و منفی می‌شوند. Integerها شامل چهار نوع data type می‌شوند:** byte, short, int, long

**floating-point: اعداد اعشاری. شامل اعداد منفی و مثبت اعشاری می‌شوند. floatin-pointها شامل چهار نوع data-type می‌شوند:** float, double, decimal.

float f1 = 123456.5F;

float f2 = 1.123456f;

double d1 = 12345678912345.5d;

double d2 = 1.123456789123456d

decimal d1 = 123456789123456789123456789.5m;

decimal d2 = 1.1234567891345679123456789123m;

**stringها در سی شارپ**

**رشته‌ها یا همان stingها در double quotes قرار می‌گیرند.**

string ch = "S";

string word = "hello";

string text = "hello world";

**نکته: آبجکت string حداکثر 2 گیگ فضا در حافظه اشغال می‌کند، که حدود یک بیلیون کاراکتر می‌شود.**

**نکته: رشته ها یا همان stringها در سی شارپ، در واقع یک آرایه ای از کاراکترها هستند.**

      string text = "hello, how are you today?";

      for (var i = 0; i < text.Length; i++)

      {

        Console.WriteLine(text[i]);

      }

**Verbatim Strings**

وقتی در یک رشته می خواهیم به خط بعد برویم یا اینکه علامت “ یا ‘ استفاده کنیم باید از scape characterها استفاده کنیم:

1. \' - single quote, needed for **character** literals.
2. \" - double quote, needed for string literals.
3. \\ - backslash.
4. \0 - Unicode **character** 0.
5. \a - Alert (**character** 7)
6. \b - Backspace (**character** 8)
7. \f - Form feed (**character** 12)
8. \n - New line (**character** 10)

حالا فرض کنید می خواهیم یک رشته که در واقع نشان دهنده یک مسیر است تعریف کنیم. مانند زیر:

string path = "c:\\projects\\project1\\folder1";

که همانطور که مشاهده می کنید کمی نا مفهوم می شود. چون به جای \ از \\ استفاده کرده ایم. برای حل این مورد در سی شارپ یک مدل خاصی از string داریم به نام verbatim string که این مورد را برطرف می کند. در verbatim string به جای استفاده از دابل بک اسلش ها یک علامت @ در ابتدای مقدار قرار می دهیم. در شکل زیر مثال بالا را به این شکل تغییر داده ایم:

string path = @"c:\projects\project1\folder1";

**تاریخ و زمان در سی شارپ – working with date and time in c#**

برای کار کردن با تاریخ و زمان در سی شارپ، باید یک آبجکت از استراکت (struct) تاریخ و زمان DateTime بسازیم.

در زیر یک آبجکت DateTime با مقدار پیش فرض (default) ایجاد کرده‌ایم:

DateTime dt = new DateTime(); // 0001-01-01T00:00:00

مقدار پیش فرض کوچکترین واحد زمانی این آبجکت است. بزرگترین واحد زمان آن هم می‌شود:

December 31, 9999 11:59:59 P.M.

برای اینکه تاریخ خودمان را اختصاص دهیم، مانند زیر عمل می‌کنیم:

DateTime dt2 = new DateTime(2020, 12, 31); //assigns year, month, day

//assigns year, month, day, hour, min, seconds

DateTime dt3 = new DateTime(2021, 12, 31, 5, 10, 20);

DateTime currentDateTime = DateTime.Now; //returns current date and time

DateTime todaysDate = DateTime.Today; // returns today's date

DateTime currentDateTimeUTC = DateTime.UtcNow;// returns current UTC date and time

DateTime maxDateTimeValue = DateTime.MaxValue; // returns max value of DateTime

DateTime minDateTimeValue = DateTime.MinValue; // returns min value of DateTime

جهت تبدیل یک رشته به یک تاریخ از توابع TryParse() و TryParseExact() استفاده می‌کنیم.

مثال زیر:

// Convert string to valid DateTime

string d = "2022/11/05"; // this is a string

DateTime dt; // this is a variable.

var myDate = DateTime.TryParse(d, out dt); // by TryParse method we convert string s to date dt.

Console.WriteLine(d); // show result in console

**Strcut in c#**

در سی شارپ مفهوم دیگری داریم که شبیه کلاس هستند به نام structure یا struct. از نظر سینتکس بسیار شبیه کلاس می باشد با این تفاوت که به جای استفاده از کلمه کلیدی class از کلمه کلیدی struct استفاده می شود. شکل زیر:

public struct rgbColor

{

public int Red;

public int Green;

public int Blue;

}

یک struct هم مانند کلاس، شامل فیلدهای مرتبط و متدها می باشد. اما تفاوت بین class و struct چیست؟ در 99 درصد مواقع ما کلاس ایجاد می کنیم و بسیار به ندرت پیش می آید که یک struct ایجاد کنیم. توصیه می شود وقتی از struct استفاده کنید که می خواهید یک کلاس خیلی کوچک تهیه کنید. مانند همین مثال rgbColor که در بالا زده شده است.

نکته: structها نمی‌توانند constructor بدون پارامتر داشته باشند. یک structor اگر می‌خواهد construcot داشته باشد، یا باید با پارامتر باشد و یا از نوع static

مثال زیر یک struct را نشان می‌دهد، همراه با یک constructor دارای پارامتر.

struct Coordinate

{

public int x;

public int y;

public Coordinate(int x, int y)

{

this.x = x;

this.y = y;

}

}

Coordinate point = new Coordinate(10, 20);

Console.WriteLine(point.x); //output: 10

Console.WriteLine(point.y); //output: 20

نکته: structها هم مانند کلاس‌ها می‌توانند پراپرتی (properties) و متد داشته باشند. مثال زیر:

struct Coordinate

{

public int x { get; set; }

public int y { get; set; }

public void SetOrigin()

{

this.x = 0;

this.y = 0;

}

}

Coordinate point = Coordinate();

point.SetOrigin();

Console.WriteLine(point.x); //output: 0

Console.WriteLine(point.y); //output: 0

نکته: structها نمی‌توانند از یک struct دیگر یا از یک کلاس دیگر ارث بری کنند.

**enum در سی شارپ**

در سی شارپ، enum یا (enumeration type) برای این استفاده می‌شود که یک اسم ثابت به یک گروه از نوع integer بدهیم. Enum ها در واقع مقادیر ثابت (constant value) را خوانا تر می کنند. (readable)

قاعده تعریف یک enum:

Enumها به طور مستقیم در namespaceها، کلاس ها و یا structureها تعریف می‌شوند. مثال زیر:

enum WeekDays

{

Monday,

Tuesday,

Wednesday,

Thursday,

Friday,

Saturday,

Sunday

}

نکته: اگر برای اعضای یک enum مقدار (value) تعریف نشود، خود کامپایلر برای هر کدام از memberهای آن یک integer در نظر می‌گیرد که از صفر شروع می شود.

enum Categories

{

Electronics, // 0

Food, // 1

Automotive = 6, // 6

Arts, // 7

BeautyCare, // 8

Fashion // 9

}

دستیابی به مقادیر یک enum - Access An Enum

enum WeekDays

{

Monday,

Tuesday,

Wednesday,

Thursday,

Friday,

Saturday,

Sunday

}

Console.WriteLine(WeekDays.Monday); // Monday

Console.WriteLine(WeekDays.Tuesday); // Tuesday

Console.WriteLine(WeekDays.Wednesday); // Wednesday

Console.WriteLine(WeekDays.Thursday); // Thursday

Console.WriteLine(WeekDays.Friday); // Friday

Console.WriteLine(WeekDays.Saturday); // Saturday

Console.WriteLine(WeekDays.Sunday); // Sunday

Conversion in enum

enum WeekDays

{

Monday,

Tuesday,

Wednesday,

Thursday,

Friday,

Saturday,

Sunday

}

Console.WriteLine(WeekDays.Friday); //output: Friday

int day = (int) WeekDays.Friday; // enum to int conversion

Console.WriteLine(day); //output: 4

**StringBuilder**

در سی شارپ stringها Immutable یا تغییر ناپذیر هستند. به این معنی که وقتی یک string تعریف می‌کنیم، این string خانه ای از حافظه را در سیستم اشغال می‌کند، اگر این string را تغییر دهیم، میاد یک خانه دیگری از حافظه را پر می‌کند با string جدید، در واقع نمیاد همان خانه ای را که قبلا اشغال کرده بود را تغییر دهد.

حالا اگر این تغییرات زیاد انجام شود، حافظه زیادی از سیستم اشغال می‌شود.

سی شارپ برای حل این مشکل StringBuilder که در نیم اسپیس System.Text را معرفی کرده است.

روش ساخت یک آبجکت StringBuilder

using System.Text; // include at the top

StringBuilder sb = new StringBuilder(); //string will be appended later

//or

StringBuilder sb = new StringBuilder("Hello World!");

بازیابی یک StringBuilder – Retrieve String from StringBuilder

در واقع چون StringBuilder یک string نیست، برای بازیابی آن باید آن را به string تبدیل کنیم که با استفاده از تابع toString() می‌توانیم این کار را انجام دهیم

StringBuilder sb = new StringBuilder("Hello World!");

var greet = sb.ToString(); //returns "Hello World!"

**namesapce:**

**با استفاده از namespaceها می‌توانیم کدهای خودمان را به فایل‌های جداگانه تبدیل کنیم، از این رو می‌توانیم کد خودمان را نظم و سامان بهتری دهیم. مثلا در مثال زیر یک فولدر با نام Models ایجاد کردیم و کلاس خودمون را در این فولدر قرار دادیم. (نام فولدر و نام کلاس یا مدل را با حروف بزرگ می‌نویسیم)**

**هر وقت فایلی ایجاد می‌کنیم ابتدا باید namespace بسازیم. نام namespace هم معمولا همان نام پروژه‌مان می‌باشد. مانند مثال زیر: ابتدا نام پروژه و سپس نام فولدر.**

 namespace HelloWorld.Models

 {

  public class Computer

  {

    public string Motherboard {get; set;}

    public int CPUCores {get; set;}

    public bool HasWifi {get; set;}

    public bool HasLTE {get; set;}

    public DateTime ReleaseDate {get; set;}

    public decimal Price{get; set;}

    public string VideoCard {get; set;}

  }

 }

**در مثال زیر یک کلاس (مدل) ایجاد می‌کنیم:**

namespace HelloWorld.Models

{

  public class Computer

  {

    //private string \_motherboard;

    // public string Motherboard

    // {

    //   get{return \_motherboard;}

    //   set{ \_motherboard = value;}

    // }

    //before dot net 6 we put ? for. but now for solve this:

    // "Non-nullable property 'Motherboard' must contain a non-null"

    // we should make a constructor for computer class.

    // public string? Motherboard { get; set; }

    public string Motherboard { get; set; }

    public int CPUCores { get; set; }

    public bool HasWifi { get; set; }

    public bool HasLTE { get; set; }

    public DateTime ReleaseDate { get; set; }

    public decimal Price { get; set; }

    public string VideoCard { get; set; }

    // this is a constructor

    public Computer()

    {

      if (Motherboard == null)

      {

        Motherboard = "";

      }

      if (VideoCard == null)

      {

        VideoCard = "";

      }

    }

  }

}

**در زیر یک آبجکت یا instance از کلاس بالا ایجاد می‌کنیم:**

      Computer myComputer = new Computer()

      {

        Motherboard = "Asus",

        HasWifi = true,

        HasLTE = true,

        ReleaseDate = DateTime.Now,

        Price = 950.15m,

        VideoCard = "Nvidia"

      };

**جهت import کردن یک کلاس در یک namespace دیگر:** using HelloWorld.Models;

**ترکیبت رشته ها با هم دیگر - String Concatenation**

**در سی شارپ با استفاده از علامت + می‌‎توان رشته ها را با یکدیگر ترکیب کرد.**

      string text = "hello" + "are you ok?";

      string txt1 = "my name is";

      string txt2 = "saeed";

      Console.WriteLine(txt1 + txt2);

**String Interpolation**

**در سی شارپ با استفاده از علامت $ و {} عملیات interpolation را انجام می‌دهیم. مثال:**

      string firstName = "James";

      string lastName = "Bond";

      string code = "007";

      string fullName = $"Mr. {firstName} {lastName}, Code: {code}";

      Console.WriteLine(fullName);

**String Method in C#**

Clone()**: یک کلون (کپی برابر اصل) از یک رشته می‌سازد.**

      string c = "hello";

      string c2 = (string)c.Clone();

      Console.WriteLine(c2);

Contains(): **این تابع چک می‌کند که آیا یک رشته شامل یک مقدار خاصی از یک رشته هست یا نیست. اکر باشد مقدار true و اگر نباشد مقدار false را بر می‌گرداند.**

      string msg = "hello everybody how are you?";

      bool s = msg.Contains("z");

      Console.WriteLine(s); //False

EndsWith(): **این متد چک می‌کند که آیا یک رشته با یک کاراکتر خاص تمام شده است یا خیر. در صورتی که درست باشد مقدار true و در غیر این صورت مقدار false را بر می‌گرداند. خروجی از نوع boolean می‌باشد.**

      string msg = "hello everybody how are you?";

      bool e = msg.EndsWith(".");

      Console.WriteLine(e); // False

**دستور مقابل آن StartsWith می‌باشد.**

ToLower() و ()ToUpper: **یک رشته را به حروف بزرگ یا کوچک تبدیل می‌کند.**

      string msg = "hello everybody how are you?";

      string e = msg.ToUpper();

      Console.WriteLine(e);

IndexOf(): **این تابع ایندکس اولین کارکتر مورد نظر را برمی‌گرداند.** **خروجی از نوع int می‌باشد.**

      string msg = "hello";

      int i = msg.IndexOf("e"); //index number start from zero

      Console.WriteLine(i); //1

LastIndexOf(): **این تابع برخلاف IndexOf که ایندکس اولین کاراکرتر را بر‌می‌گرداند، از آخر ایندکس کاراکتر مورد نظر را بر می‌گرداند.**

      string msg = "hello world";

      int i = msg.LastIndexOf('l');

      Console.WriteLine(i); // 9

Insert(): **این تابع دوتا ورودی (arguman) می‌گیرید. اولی از نوع int که ایندکس را مشخص می‌کند و دومی رشته‌ای که باید در ایندکس مورد نظر insert شود.**

**نکته: رشته مورد نظر را قبل از ایندکس مورد نظر insert می‌کند.**

      string msg = "hello world";

      string newMsg = msg.Insert(5, " saeed");

      Console.WriteLine(newMsg);

Length: **طول یک رشته را برمی‌گرداند.**

      string msg = "hello world";

      int i = msg.Length;

      Console.WriteLine(i); // 11

Replace(): **این تابع دوتا آرگومان از نوع string می‌گیرد. در رشته مورد نظر آرگومان دوم را جایگزین آرگومان اول می‌کند.**

      string txt = "Hello world";

      string newTxt = txt.Replace("world", "saeed");

      Console.WriteLine(newTxt);

Substring(): **این متد یا تابع دوتا آرگومان می‌گیرد، آرگومان اول ایندکس شروع و آرگومان دوم طول که اجباری نیست.**

Substring(int startIndex, int length)

* startIndex - the beginning index of the substring
* length - (optional) - length of the substring

      string txt = "Hello world";

      string newTxt = txt.Substring(0, 5);

      Console.WriteLine(newTxt); // Hello

Trim(): **فضای خالی ابتدا وانتهای یک رشته را حذف می‌کند.**

**تمرین:**

**برنامه ای بنویسید که یک آدرس ایمیل از کاربر دریافت کند و قسمت دومین آن را بر گرداند.**

      Console.WriteLine("enter your email:");

      string email = Console.ReadLine();

      // find index of @

      int i = email.IndexOf('@');

      string d = email.Substring(i + 1);

      Console.WriteLine(d);

**برنامه ای بنویسید که یک عدد از کاربر دریافت کند و قدر مطلق عدد را نشان دهد. (عدد منفی مثب شود)**

      Console.WriteLine("enter a number:");

      int num = int.Parse(Console.ReadLine());

      if (num >= 0)

      {

        Console.WriteLine(num);

      }

      else

      {

        Console.WriteLine(num \* -1);

      }

**برنامه بنویسید که یک عدد از کاربر دریافت کند و مشخص کند عدد زوج است یا فرد.**

      Console.WriteLine("enter a number:");

      int num = int.Parse(Console.ReadLine());

      if (num % 2 == 0)

      {

        Console.WriteLine("even");

      }

      else

      {

        Console.WriteLine("Odd");

      }

**برنامه ای بنویسید که مشخص کند عددی که از کاربر دریافت کرده است مثبت است یا منفی.**

      Console.WriteLine("enter a number:");

      int num = int.Parse(Console.ReadLine());

      if (num >= 0)

      {

        Console.WriteLine("positive");

      }

      else

      {

        Console.WriteLine("negative");

      }

**برنامه ای بنویسید که دو عدد از کابر دریافت کند وچک کند که آیا اعداد برابر هستند یا خیر.**

      Console.WriteLine("enter a number:");

      int num = int.Parse(Console.ReadLine());

      Console.WriteLine("enter a number:");

      int num2 = int.Parse(Console.ReadLine());

      if (num == num2)

      {

        Console.WriteLine("they are equal");

      }

      else

      {

        Console.WriteLine("not equal");

      }

**برنامه‌ای بنویسید که قسمت username یک آدرس ایمیل را برگرداند.**

      string mail = "saeed@gmail.com";

      int i = mail.IndexOf('@');

      string user = mail.Substring(0, i);

      Console.WriteLine(user);

**برنامه ای بنویسید که شاخص توده بدنی را محاسبه کند و نتیجه را گزارش دهد.**

فرمول:

وزن به کیلوگرم تقسیم بر قد بر حسب متر به توان 2

چناچه BMI زیر ۱۸.۵ باشد خیلی لاغر است عددی بین ۱۸.۵ و ۲۴.۹ باشد خوب و مناسب است. بین ۲۵ و ۲۹.۹ باشد مقداری اضافه وزن دارد. بین ۳۰ و ۳۴.۹ باشد چاق است. بالای ۳۵ باشد چاقی مفرط است.

      Console.WriteLine("enter weight:");

      double num = double.Parse(Console.ReadLine());

      Console.WriteLine("enter height:");

      double num2 = double.Parse(Console.ReadLine());

      double result = (num) / Math.Pow(num2, 2);

      double result2 = Math.Round((num) / Math.Pow(num2, 2), 2);

      if (result < 18.5)

      {

        Console.WriteLine("too much thin");

      }

      else if (result > 18.5 || result < 24.9)

      {

        Console.WriteLine("it is ok and suitable");

      }

**Ternary Operator**

**به جای استفاده از if condition می‌شود از ?: استفاده کرد. در واقع حالت کوتاه شده‌ای برای دستور شرطی if است، که بهش می‌گن ternary operator یا conditional operator. نحوه نوشتن یا همان syntax آن به صورت زیر است:**

condition ? statement 1 : statement 2

**مثال:**

int x = 20, y = 10;

var result = x > y ? "x is greater than y" : "x is less than y";

Console.WriteLine(result); // x is greater than y

**Switch Statement**

**دستور switch statement را می‌شود به جای دستور if else استفاده کرد. وقتی می‌خواهیم یک مقداری را در مقابل حالت‌های مختلف بررسی کنیم. نحوه نگارش یا syntax این دستور به شکل زیر است:**

switch(match expression/variable)

{

case constant-value:

statement(s) to be executed;

break;

default:

statement(s) to be executed;

break;

}

**مثال:**

      int a = 5;

      switch (a)

      {

        case 5:

          Console.WriteLine("value of a is 5");

          break;

        case 10:

          Console.WriteLine("value of a is 5");

          break;

        case 15:

          Console.WriteLine("value of a is 15");

          break;

        default:

          Console.WriteLine("unknown");

          break;

      }

    }

**نکته: قسمت default در یک switch statement اجباری نیست.**

**نکته: از چندتا case هم می‌شود در یک دستور switch استفاده کرد. مثال:**

      int a = 5;

      switch (a)

      {

        case 1:

          Console.WriteLine("a = 1");

          break;

        case 2:

        case 5:

          Console.WriteLine("a is 2 or a is 5");

          break;

      }

**حلقه‌ی for یا همان for Loop**

**حلقه‌ی for، تا وقتی که شرط برقرار باشد، حلقه تکرار می‌شود. syntax حلقه for به صورت زیر می‌باشد:**

for (initializer; condition; iterator)

{

//code block

}

**مثال:**

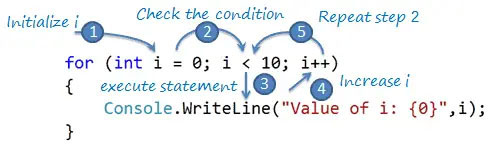
      for (int i = 0; i < 10; i++)

      {

        Console.WriteLine(i);

      }

**خروجی حلقه‌ی بالا می‌شود اعداد صفر الی 9.**

****

**نکته: در حلقه for سه تا بخشی که داخل پرانتز می‌نویسیم اجباری نیست. به عنوان مثال حلقه زیر تا ابد اجرا می‌شود:**

for ( ; ; )

{

Console.Write(1);

}

**نکته: در حلقه for قسمت iterator یا همان تکرار شونده، می‌توان به صورت کاهشی هم انجام داد. مثال زیر اعداد 10 تا یک را چاپ می‌کند:**

      for (int i = 10; i > 0; i--)

      {

        Console.WriteLine(i);

      }

**خروج از حلقه for**

**با تعیین کردن شرط می‌توان از حلقه خارج شد. مثال زیر:**

      for (int i = 0; i < 10; i++)

      {

        if (i == 8)

        {

          break;

        }

        Console.WriteLine(i);

      }

**تمرین:**

**با استفاده از حلقه for برنامه‌ای بنویسید که مجموع اعداد یک تا 10 را چاپ کند:**

      int n = 0;

      for (int i = 0; i <= 10; i++)

      {

        n = n + i;

        // n += i;

        Console.WriteLine(i);

      }

      Console.WriteLine(n);

**حلقه while loop**

**حلقه while loop هم مانند حلقه for عمل می‌کند. تا وقتی که شرط صحیح باشد، حلقه ادامه دارد. ساختار کلی آن به صورت زیر است:**

While(*condition*)

{

//code block

}

**مثال: اعداد یک تا ده تا با استفاه از حلقه while loop چاپ کنید.**

int i = 0;

while (i < 10) // شرط

{

Console.WriteLine("i = {0}", i);

i++; // افزایش دهنده

}

**متدها در سی شارپ – Method**

**متد چیست: متدها مجموعه‌ای از کدها هستند که در کنار یکدیگر وظیفه‌ای را انجام می‌دهند.**

* **متدها قبل از استفاده باید تعریف (ایجاد) شده باشند. (define the method)**
* **متدها برای اجرا شدن حتما باید صدا زده شوند. (call the method)**
* **به متد، تابع هم می‌گویند. (function)**
* **متدها می‌توانند دارای پارامتر (parameters) باشند و یا اینکه بدون پارامتر باشند. (به متغیرهایی که در داخل پرانتز یک متد نوشته می‌شوند می‌گویند پارامتر)**
* **هر متد باید دارای یک نام باشد. ترجیح این است که حرف اول نام یک متد با حرف بزرگ باشد.**
* **اگر متدی که نوشتیم مقداری را return نمی‌کند قبل از نام متد باید از کلمه void استفاده کنید.**
* **اگر متد بدون void باشد باید نوع متد مشخص باشد. مثلا int یا string یا double و...**
* **اگر می‌خواهید متد را بدون تعریف آبجکت استفاده کنید، قبل از نام متد کلمه static را استفاده کنید.**
* **با استفاده از علامت مساوی = می‌توان مقدار پیش فرض برای یک پارامتر مشخص کرد.**
* **می‌توان متد هم نام ایجادکرد اما باید پارامتر آن متفاوت باشد. به این حالت می‌گویند overloading method**

**چرا از متدها (توابع) استفاده می‌کنیم: اگر می‌خواهیم کدی را بارها و بارها استفاده کنیم، آن کد را در قالب متد می‌نویسیم و هر بار که نیاز داشتیم متد مورد نظرمان را فرخوانی می‌کنیم.**

**روش ایجاد یک متد (syntax): در مثال زیر یک متد با نام Greeting تعریف کردیم.**

    void Greeting()

    {

      Console.WriteLine("Hello from Greeting method");

    }

**در مثال بالا یک متد ساختیم با نام Greeting. چون این متد مقداری را return یا برگشت نمی‌کند از کلمه void قبل از نام متد استفاده می‌کنیم.**

**مثال دیگری از تعریف متد:**

    string Test()

    {

      return "Hello from Test method";

    }

**در مثال بالا، چون این متد مقداری را return می‌کند، بنابراین در نام متد از کلمه void استفاده نکردیم. در این حالت باید نوع بازگشت متد (data type) را مشخص کنیم که در مثال بالا از نوع string است.**

**تمرین:**

**متدی با نام Hello بنویسید که عبارت Hello World را نمایش دهد.**

    void Hello()

    {

      Console.WriteLine("Hello World");

    }

**متدی با نام Greeting بنویسید که دارای پارامتری با نام name باشد. که بعد از دریافت نام کاربر کلمه Hello را قبل از نام کاربر اضافه کند.**

    static void Greeting(string fname)

    {

      Console.WriteLine("Hello " + fname);

    }

**جهت فراخوانی متد (صدا زدن):**

      Greeting("David");

**به چی می‌گن پارامتر (Parameter) و به چی می‌گن آرگومان (Argument): مثلا در مثال بالا به fname می‌گویند پارامتر و به David می‌گویند آرگومان.**

**متدی بنویسید که دو تا مقدار دریافت کند و جمع دو عدد را برگرداند.**

    static int SumNumbers(int a, int b)

    {

      int c = a + b;

      return c;

    }

**صدا زدن متد بالا:**

    Console.WriteLine(SumNumbers(5, 10));

**متدی بنویسید که دوتا آرگومان از نوع عدد دریافت کند و عدد بزرگتر را برگرداند.**

    static int BiggerNum(int a, int b)

    {

      if (a > b)

      {

        return a;

      }

      else

      {

        return b;

      }

    }

**در زیر متد را صدا زدیم:**

      int big = BiggerNum(7, 3);

      Console.WriteLine(big); // 7

**متد (تابع) بازگشتی - (Recursive Method):**

**یک متد می‌تواند خودش را داخل خودش صدا بزند. به این حالت میگن بازگشتی یا recursion**

**تمرین:**

**با استفاده از متد بازگشتی، متدی بنویسید که یک عدد به عنوان آرگومان دریافت کند و فاکتوریل عدد را حساب کند.**

**فرمول فاکتوریل: n!=n×(n−1)×(n−2)×...×1**

**5!=5×4×3×2×1=120**

    static int Factorial(int num)

    {

      int result;

      if (num == 1)

      {

        return 1;

      }

      else

      {

        result = Factorial(num - 1) \* num;

        return result;

      }

    }

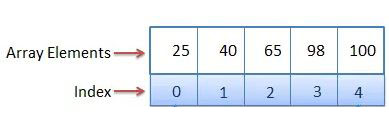
**متد را صدا زدیم**:

      Console.WriteLine(Factorial(5)); // 120

**آرایه‌ها – array**

**آرایه (array) چیست: آرایه متغیری است که به جای نگهداری یک مقدار می‌تواند چندین مقدار را درون خودش نگهداری ‌کند. طول این مقادیر باید مشخص باشد، همچنین این مقادیر باید از یک نوع (data type) باشند.**

**نکته: مقادیر یا المنت‌ها در آرایه‌ها دارای ایندکس (index) می‌باشند که از صفر شروع می‌شود.**

****

***در متغیرها، فقط یک مقدار نگهداری می‌شود، در حالی که در آرایه‌ها چندین مقدار نگهداری می‌شوند.***

**تعریف آرایه – (declare an array): برای تعریف یک آرایه ابتدا نوع داده‌ای آرایه را همراه با علامت [] مشخص می‌کنیم و سپس نام آرایه.**

datatype[] arrayName;

**در مثال زیر یک آرایه با نام cities از نوع string تعریف کردیم:**

int[] number = new int[4] { 20, 30, 40, 50 };

**در مثال بالا یک آرایه از نوع int تعریف کردیم با طول ثابت 4 مقدار**.

string[] cities = new string[] { "Tehran", "Mashad", "Shiraz" };

**در مثال بالا طول آرایه را مشخص نکردیم. در این حالت کامپایلر سی شارپ بر اساس تعداد مقادیر وارد شده، طول آرایه را در نظر می‌گیرد.**

**تعریف آرایه به یک روش دیگر:**

int[] num = { 1, 2, 3 };

**نکته: اگر می‌خواهیم آرایه‌ای تعریف کنیم و بعدا به آن مقدار دهیم، حتما باید طول آن را مشخص کنیم. در غیر اینصورت آرایه از نظر سی شارپ معتبر نیست. مثلا مثال زیر نا معتبر است:**

int[] number = new int[];

**در این حالت حتما باید طول آرایه را مشخص کنیم. مثال بالا را به این شکل تصحیح می‌کنیم:**

int[] number = new int[5];

**نکته: با استفاده از کلمه کلیدی var هم ‌می‌توانیم آرایه تعریف کنیم. مثال زیر:**

var cars = new string[] { "pride", "benz", "bmw" };

**نکته: برای تعریف آرایه می‌توان ابتدا نام آرایه را مشخص کرد و بعدا آن را مقدار دهی (initialize) کرد. مثال زیر:**

int[] numbers;

numbers = new int[3];

**دسترسی به مقادیر یا المنت‌های آرایه**

**برای دسترسی به المنت‌های یک آرایه باید شماره ایندکس آن را مشخص کنیم. مثال زیر:**

string[] cars = new string[] { "pride", "benz", "bmw", "prado" };

Console.WriteLine(cars[1]); // benz

**دسترسی به المنت‌های یک آرایه با استفاده از حلقه for**

for (int i = 0; i < cars.Length; i++)

{

 Console.WriteLine(cars[i]);

}

**دسترسی به المنت‌های یک آرایه با استفاده از foreach**

foreach (var item in cars)

{

  Console.WriteLine(item);

}

**تغییر المنت آرایه:**

cars[0] = "mvm";

**تمرین:**

**برنامه ای بنویسید که مجموع اعداد یک آرایه عددی را حساب کند.**

int[] numbers = new int[] { 10, 20, 30, 40 };

int s = 0;

for (int i = 0; i < numbers.Length; i++)

{

  // s += numbers[i];

  s = s + numbers[i];

}

Console.WriteLine(s);

**برنامه ای بنویسید که اعداد زوج و فرد را در یک آرایه عددی جدا کند.**

int[] numbers = new int[] { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };

for (int i = 0; i < numbers.Length; i++)

{

  if (numbers[i] % 2 == 0)

  {

    Console.WriteLine("even: " + numbers[i]);

  }

}

for (int i = 0; i < numbers.Length; i++)

{

  if (numbers[i] % 2 != 0)

  {

    Console.WriteLine("odd: " + numbers[i]);

  }

}

**آرایه‌های دو بعدی (Two-Dimensional Arrays):**

**برای تعریف یک آرایه دو بعدی، باید در علامت [] یک کاما ( , ) قرار دهید. برای مثال در زیر یک آرایه دو بعدی ساختیم:**

      int[,] num = new int[,]

      {

        {1,2,3},

        {4,5,6}

      };

      Console.WriteLine(num[0, 0]); // 0

**دسترسی به آرایه دو بعدی با استفاده از foreach.**

foreach (var item in num)

{

  Console.WriteLine(item);

}

**مثال دیگری از آرایه دو بعدی. این آرایه سه ردیف دارد و طول هر ردیف برابر است با 2**

int[,] test = new int[3, 2]

{

 {4,5},

 {7,8},

 {10,11}

};

**آرایه‌های سه بعدی (Three-Dimensional Arrays):**

int[,,] arr = new int[2, 2, 3]

 {

  { {1,2,3}, {4,5,6} },

  { {10,11,12}, {13,14,15} }

 };

**نکته:** **در سی شارپ تا آرایه‌ی 32 بعدی می‌توان ایجاد.**

**آرایه در آرایه (jagged arrays):**

**در سی شارپ می‌توان آرایه ای درون یک آرایه دیگر تعریف کرد که به آن jagged array می‌گویند. مثال:**

**عددی که براکت اول قرار می‌گیرد تعیین کننده سایز آرایه و عدد که در براکت دوم قرار می‌گیرد نشان دهنده بعد آرایه است.**

int[][] jarr = new int[2][];

jarr[0] = new int[3] { 2, 5, 7 };

jarr[1] = new int[4] { 3, 8, 9, 1 };

**Structure**

**توسط structure می‌توانیم یک متغیر تعریف کنیم که داده‌های به هم مرتبط را که data type مختلفی دارند را نگهداری کند. مثلا حالت زیر را در نظر بگیرید، اطلاعات کتاب را به صورت یک struct تعیین تعریف می‌کنیم:**

**Title**

**Author**

**Subject**

**Book ID**

**نحوه تعریف structure (defining a Structure):**

struct Book

{

  public string title;

  public string author;

  public string subject;

  public int book\_id;

}

**در مثال بالا یک struct با نام Book ایجاد کردیم.**

**برای اینکه از struct ساخته شده، یک آبجکت بسازیم، هم ‌می‌توان از کلمه new استفاده کرد و هم استفاده نکرد.**

Book book; // **ساخت یک آبجکت از استراکت**

Book book1 = new Book(); // **ساخت یک آبجک از استراکت**

book.title = "hello"; // **مقدار دهی**

Console.WriteLine(book.title); // **نمایش عنواین**

**بعضی از ویژگی های structureها**

* **یک structure در داخل خود می‌تواند method، field، indexer، property، و event داشته باشد.**
* **Structureها نمی‌توانند از یک دیگر یا از یک کلاس دیگر ارث بری کنند.**
* **اعضای (member) یک structure نمی‌توانند به صورت protect، virtual یا abstract تعیین شوند.**

**Enum – Enumeration Type**

**اختصاص عدد به یک سری نام های ثابت. enumها داخل namespaceها، کلاس‌ها و یا structureها تعریف می‌شوند.**

**مثال:**

enum Season

{

  spring , summer, fall, winter

}

**نحوه دستیابی:**

Console.WriteLine(Season.spring); // spring

int x = (int)Season.fall;

foreach (var item in Enum.GetNames(typeof(Season)))

{

  Console.WriteLine(item);

}

**تغییر عدد enum:**

enum Season

{

  spring = 1, summer, fall, winter

}

**کلاس - class**

**کلاس‌ها در واقع نوعی طرح کلی از یک چیزی هستند. مثلا یک طرح کلی از یک ماشین را در نظر بگیرید، سپس از آن کلاس که همون طرح کلی ما می‌باشد نمونه‌هایی (object) ایجاد می‌کنیم. objectها در واقع نمونه (instance) هایی از کلاس‌ها هستند.**

**یک کلاس می‌تواند شامل یک یا بیشتر از یک constructor، field, method, property, delegate و event که به آن‌ها member می‌گویند باشد. یک کلاس و memberهای آن می‌توانند access modifier خودشان را مانند public، private، protected و internal برای محدود کردن دستیابی به آن از از دیگر قسمت‌های برنامه را داشته باشد.**

**در زیر یک کلاس تعریف کردیم که دارای یک فیلد از نوع id است:**

**فیلدها در واقع همان متغیرها (variable) هستند. به طور کلی فیلدها باید access modifier از نوع private داشته باشند.**

class Students

{

  public int id;

}

**Property**

**پراپرتی میاد یک فیلد از نوع private را encapsulate می‌کند و با استفاه از دستورات setter و getter مقدار را به فیلد اختصاص و بازیابی می‌کند. مثال زیر:**

class Students

{

  private int id;

  public int StudentId

  {

    get { return id; }

    set { id = value; }

  }

}

**همچنین می‌توانیم به get و set هم شرط هم هم اضافه کرد. مانند زیر:**

class Students

{

private int id;

public int StudentId

{

  get { return id; }

  set

  {

    if (value > 0)

    {

      id = value;

    }

  }

}

}

**Auto-implemented Property**

**از سی شارپ نسخه 3 به بعد تعریف property ها راحت تر شد. البته اگر نمی‌خواهید شرط برای propertyها قرار دهید.**

**در زیر مثال بالا را به این شکل راحت تر می‌نویسیم:**

  class Students

  {

    public int StudentId { get; set; }

    public string FirstName { get; set; }

    public string LastName { get; set; }

  }

**می‌توان در کلاس، متد هم نوشت. مثلا:**

  class Students

  {

    public int StudentId { get; set; }

    public string FirstName { get; set; }

    public string LastName { get; set; }

    public string FullName()

    {

      return FirstName + " " + LastName;

    }

  }

**حالا از کلاس بالا یک آبجکت می‌سازیم و مقدار دهی می‌کنیم. مانند زیر:**

Students st1 = new Students();

st1.FirstName = "Akbar";

st1.LastName = "Asghari";

string fullname = st1.FullName();

Console.WriteLine(fullname);

**Constructor**

**constructorها متدهایی هستند که وقتی یک instance از کلاس می‌سازیم، به طور اتوماتیک اجرا می‌شوند.**

* **نام یک constructor باید دقیقا مانند نام کلاس باشد.**
* **یک constructor می‌تواند به صورت public، private و یا protected باشد.**
* **یک constructor نمی‌تواند مقداری را return کند. بنابراین نباید یک return type داشته باشد.**
* **یک کلاس می‌تواند شامل چندین constructor باشد که هر کدام از این constructorها پارامتر مختلفی داشته باشند. اما فقط می‌تواند یک constructor بدون پارامتر داشته باشد.**
* **اگر برای کلاس خودمان هیچ constructor تعریف نکنیم، سی شارپ خودش یک constructor داخلی ایجاد می‌کند.**

  class Car

  {

    public string Name { get; set; }

    public Car()

    {

      Name = "Benz";

    }

  }

 Car car1 = new Car();

 // string c = car1.Name = "pride";

 Console.WriteLine(car1.Name); // Benz

**ارث بری – inheritance**

  class Automobile : Car

  {

    public int AutoId { get; set; }

  }

**Sealed**

**اگر نمی‌خواهید که کلاس‌‎های دیگر بتوانند از کلاس مورد نظر شما ارث بری کنند باید آن را sealed کنید. مثال زیر:**

  sealed class Bike

  {

  }

**Polymorphism**

**وقتی در کلاس base متدی می‌سازیم و در کلاس هایی که از کلاس base ارث بری کردند هم متدی هم نام با متد کلاس base می‌سازیم. وقتی که می‌خواهیم متدی که در کلاس derived شده است با متدی که در کلاس base هست جایگزین شود. باید متدی که در کلاس base هست رو از نوع virtual قرار دهیم و متد derived شده را از نوع override قرار دهیم.**

**مثال زیر:**

  class Animal

  {

    public virtual void animalSound()

    {

      Console.WriteLine("each animal make own sound");

    }

  }

  class Cow : Animal

  {

    public override void animalSound()

    {

      Console.WriteLine("the Cow make MOW");

    }

  }

Animal b = new Cow();

b.animalSound(); // the Cow make MOW

**Abstraction**

**اگر کلاس یا متدی را از نوع abstract قرار دهیم، نمی‌توانیم از آن آبجکتی درست کنیم، فقط می‌توانیم یک کلاس دیگه‌ای درست کنیم که از کلاسی که abstract شده است ارث بری کنیم.**

**معمولا از abstract برای موارد امنیتی استفاده می‌شود.**

**Interface**

**اینترفیس‌ها شبیه کلاس‌ها هستند با این تفاوت که فقط می‌توانند شامل متد و پراپرتی باشند، آن هم بدون body. اینترفیس‌ها از نوع abstract هستند و برای استفاده از آن‌ها باید ازشون ارث بری کرد. نمونه زیر:**

  interface ICar

  {

    void run();

  }

  class Car : ICar

  {

    public void run()

    {

      Console.WriteLine("the car run");

    }

  }

Car c = new Car();

c.run();

* **مانند abstract classها، از interfaceها نمی‌توان به صورت مستقیم آبجکت ساخت.**
* **متدهایی که در interfaceها می‌سازیم، بدنه ندارند. برای ساخت بدنه برای این متدها باید در کلاسی که از این interface ارث بری کرده بدنه ساخت.**
* **اینترفیس‌ها می‌توان شامل متدها و پراپرتی‌ها باشند اما نمی‌توانند شامل فیلد (متغیر) باشند.**
* **اینترفیس‌ها نمی‌توانند constructor داشته باشند.**

**Generic**

**Generic: به معنای عمومی است، یعنی محدود به یک نوع خاص از داده نمی‌شود. برای تعریف یک generic، به این شکل عمل می‌کنیم که در داخل علامت <> که بعد از نام generic قرار می‌دهیم، نوع داده (Data type) را مشخص می‌کنیم. مثال:**

**TypeName<T>**

**به جای T دیتا تایپ مورد نظر خودمان را قرار می‌دهیم.**

class DataStore<T>

{

  private T[] \_data = new T[10];

  public void AddorUpdate(int index, T item)

  {

    if (index >= 0 && index <= 10)

    {

      \_data[index] = item;

    }

  }

  public T GetData(int index)

  {

    if (index >= 0 && index <= 10)

    {

      return \_data[index];

    }

    else

    {

      return default(T);

    }

  }

}

      DataStore<string> d1 = new DataStore<string>();

      d1.AddorUpdate(1, "hello");

      Console.WriteLine(d1.GetData(1));

**arrayList**

**arrayList ها از نوع non-generic هستند که سایزشون به طور اتوماتیک قابل افزایش است. در یک arraylist می‌توانیم داده هایی از نوع‌های مختلف قرار دهیم. داده هایی که نه نوع آن‌ها را می‌دانیم و نه سایز آن‌ها را. مثال:**

      ArrayList ar = new ArrayList();

      ar.Add("hi");

      ar.Add(2);

      foreach (var item in ar)

      {

        Console.WriteLine(item);

      }

**با استفاده از متد AddRange(ICollection c) می‌توانیم یک** [**Array**](https://www.tutorialsteacher.com/csharp/array-csharp)**,**[**HashTable**](https://www.tutorialsteacher.com/csharp/csharp-hashtable)**,**[**SortedList**](https://www.tutorialsteacher.com/csharp/csharp-sortedlist)**, ArrayList, BitArray,**[**Queue**](https://www.tutorialsteacher.com/csharp/csharp-queue)**, and**[**Stack**](https://www.tutorialsteacher.com/csharp/csharp-stack)

**را به arraylist اضافه کنیم.**

**مثال:**

      ArrayList ar = new ArrayList();

      ArrayList ar1 = new ArrayList() { 1, "Bill", " ", true, null };

      int[] arr = { 100, 200, 300, 400 };

      Queue q = new Queue();

      q.Enqueue("Hello");

      q.Enqueue("world");

      ar.AddRange(ar1);

      ar.AddRange(arr);

      ar.AddRange(q);

      foreach (var item in ar)

      {

        Console.WriteLine(item);

      }

**Stack**

**stackها یکی از انواع collectionها هستند که داده ها به صورت LIFO یا همان (Last In First Out) یعنی به ترتیب عکس ورود آن‌ها ذخیره می‌کنند. سی شارپ هم stack از نوع generic پشتیبانی می‌کند و هم از نو** non-generic

**Generic: Stack<T>**

**non-Generic: Stack**

* **در stack المنت‌ها به ترتیب عکس ورود ذخیره می‌شوند. یعنی از آخر به اول**
* **برای اضافه کردن یک المنت به یک stack از متد push() استفاده می‌کنیم**
* **برای بازیابی المنت‌ها از stackها از متدهای Pop() و Peek() استفاده می‌کنیم.**

**مثال:**

Stack<int> st = new Stack<int>();

st.Push(1);

st.Push(2);

st.Push(3);

foreach (var item in st)

{

  Console.WriteLine(item); // 3,2,1

}

**می‌توانیم یک stack از یک آرایه تعریف کنیم. مثال زیر:**

      int[] ar = new int[] { 1, 2, 3, 4 };

      Stack<int> st = new Stack<int>(ar);

      foreach (var item in st)

      {

        Console.WriteLine(item);

      }

**Queue**

**Queueها مانند stackها هستند با این تفاوت که دقیقا برعکس stackها هستند. یعنی به شکل FIFo (First in First Out) هستند یعنی به ترتیب ورود.**

**برای اضافه کردن المنت به queueها از متد Enqueue() استفاده می‌کنیم و برای بازیابی اطلاعات از متد Dequeue() و Peek() استفاده می‌کنیم.**

**مثال:**

Queue<int> q = new Queue<int>();

q.Enqueue(1);

q.Enqueue(2);

q.Enqueue(3);

foreach (var item in q)

{

  Console.WriteLine(item);

}

**List**

**List : listها یک collection هستند که داده‌هایی از یک نوع (same-type) را به صورت یکی پس از دیگری ذخیره می‌کنند.**

**نحوه تعریف List:**

**برای تعریف و مقدار دهی به یک List می‌توانیم از روش‌های مختلف استفاده کنیم:**

// 1

List<int> list = new List<int>();

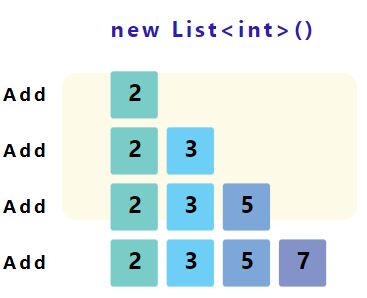
// 2

var number = new List<int>();

number.Add(2);

// 3

var number2 = new List<int>() { 2, 3, 5 };



**نکته: لیست‌ها هم مانند آرایه‌ها دارای ایندکس هستند که از صفر شروع می‌شود.**

**مثال: با استفاده از حلقه for المنت‌های یک لیست را به صورت از انتها به ابتدا بازیابی کنید:**

for (var i = number2.Count - 1; i >= 0; i--)

{

  Console.WriteLine(number2[i]);

}

**تمرین: متدی بنویسید که یک لیست را به عنوان آرگومان دریافت کند، و اگر در لیست دریافتی عدد 300 وجود داشت true برگرداند در غیر این صورت false برگرداند.**

static bool Contain300(List<int> list)

{

  foreach (var item in list)

  {

    if (item == 300)

    {

      return true;

    }

  }

  return false;

}

// this is list that contain 300

var number = new List<int>() { 100, 200, 300 };

// call the method. pass the number list as a argument

Contain300(number);

Console.WriteLine(Contain300(number));

**بازیابی داده‌های یک List به روش دیگر:**

var animals = new List<string>() { "cat", "dog", "bird" };

animals.ForEach(a => Console.WriteLine(a));

**متد TrueForAll: این متد چک می‌کند که آیا تمام المنت‌های یک List برای شرط مورد نظر true باشند. مثال:**

var numbers = new List<int>() { 10, 20, 30 };

if (numbers.TrueForAll(el => el < 40))

{

  Console.WriteLine("all the elements are less than 40");

}

**اضافه کردن یک جدا کننده:**

var colors = new List<string>() { "red", "blue", "green" };

string text = string.Join(",", colors);

Console.WriteLine(text);

**Dictionary**

**Dictionary<Tkey, TValue>: دیکشنرها یکی از انواع collectionها هستند از نوع generic که داده‌ها را به صورت key و value بدون ترتیب مشخصی ذخیره می‌کنند.**

* **در دیکشنرها کلیدها (keys) باید یونیک باشند.**
* **مقادیر یا همان valueها در دیشکنری، می‌توانند تکراری و یا null باشند.**
* **برای دستیابی به یک value باید از طریق کلید (key) آن اقدام کنیم.**

**تعریف Dictionary: در زیر تعریف یک دیشکنری به روش‌های مخلتف نشان داده شده است:**

// 1

Dictionary<int, string> dic = new Dictionary<int, string>();

dic.Add(1, "number one");

// 2

var dic2 = new Dictionary<string, int>();

dic2.Add("first", 1);

// 3

var dic3 = new Dictionary<string, int>()

{

  {"one", 1},

  {"two", 2},

  {"three", 3},

};

**دستیابی به مقادیر یک دیکشنری:**

**با استفاده از foreach:**

Dictionary<int, string> dic = new Dictionary<int, string>();

dic.Add(1, "number one");

dic.Add(2, "number two");

dic.Add(3, "number three");

dic.Add(4, "number four");

foreach (var item in dic)

{

  Console.WriteLine("Key: {0}, value: {1}", item.Key, item.Value);

}

**متد TryGetValue:**

Dictionary<int, string> dic = new Dictionary<int, string>();

dic.Add(1, "number one");

dic.Add(2, "number two");

string result;

if (dic.TryGetValue(1, out result))

{

  Console.WriteLine(result);

}

**متد GetValueOrDefault: این متد یک کلید می‌گیرد و value کلید مورد نظر را برمی‌گرداند.**

var dic = new Dictionary<int, string>();

dic.Add(1, "one");

Console.WriteLine(dic.GetValueOrDefault(1));

**اگر کلیدی که وارد شده است موجود نباشد عدد صفر را برمی‌گرداند. در صورتی که از نوع Int باشد.**

**متد TryAdd: این متد قبل از اینکه یک key و value اضافه کند، چک می‌کند که key تکراری نباشد. در صورتی که key تکراری باشد key و value را اضافه نمی‌کند. به طور کلی از ورود کلید تکراری جلوگیری می‌کند. دیتا تایپ این متد از نوع Boolean می‌باشد.**

dic.TryAdd(3, "ss");

**SortedList**

**SortedList: SortedListها از نوع collection‌ها هستند و همانند dictionaryها داده‌ها را به صورت key و value ذخیره می‌کنند که این داده‌‌ها بر اساس key یا کلیدها مرتب (sort) می‌شوند.**

**SortedList<TKey, TValue> و SortedList**

**سی شارپ هم generic و هم non-generic را در SortedListها پشتیبانی می‌کند. اما پیشنهاد می‌شود که از نوع generic استفاده شود. ( SortedList<TKey, TValue> ) چون سرعت بالاتری دارد و احتمال خطا کمتر است.**

* **برای استفاده از SortedListها باید از namespace زیر استفاده کنیم. System.Collection.Generic**
* **SortedListها المنت ها را به شکل مرتب شده ذخیره می‌کنند که binary search در این مدل فعال می‌شود. و برای نوشتن سرچ نیازی به کد نویسی نیست.**
* **SortedListها نسبت به dictionaryها کارایی (performance) ضعیف تری دارند.**

**تعریف SortedList: در زیر تعریف SortedList به روش های مختلف نشان داده شده است.**

// 1

SortedList<int, string> numberNames = new SortedList<int, string>();

// add element

numberNames.Add(3, "three");

numberNames.Add(1, "one");

numberNames.Add(2, "two");

// 2

var sorted = new SortedList<string, int>();

sorted.Add("zebra", 3);

sorted.Add("cat", 1);

sorted.Add("dog", 2);

**متدها و دستیابی‌ها**

// Use ContainsKey

bool c = sorted.ContainsKey("dog");

// Use TryGetValue

int val;

if (sorted.TryGetValue("cat", out val))

{

  Console.WriteLine(val);

}

// Loop over SortedList data

foreach (var item in sorted)

{

  Console.WriteLine(item);

}

// Get index of key and then index of value.

int index = sorted.IndexOfKey("cat"); // 0. first it will be sort then index will be show.

int index2 = sorted.IndexOfValue(2); // 1

// Display Count.

int count = sorted.Count; // 3

**Hashtable**

**Hashtableها از نوع collectionهای non-generic هستند که داده‌ها را به صورت key و value ذخیره می‌کنند. Hashtableها جستجو را سریعتر می‌کنند چون به هر key یک کد hash اختصاص می‌دهند. به عبارتی دگیر، برای جستجو در المنت‌ها از کد hash که به کلید مورد نظر داده اند استفاده می‌کنند، همین عمل باعث بالارفتن سرعت جستجو می‌شود.**

**Hashtableها تحت namespace زیر کار می‌کنند:** using System.Collections;

**تعریف و دستیابی به hashtable**

// 1

Hashtable hashtable = new Hashtable();

hashtable.Add("bird", 1);

hashtable.Add("cat", 2);

hashtable.Add("dog", 10);

// 2

var cities = new Hashtable(){

  {"IR", "Teh, Shz, MSHD"},

  {"US", "NYK, LosAngles, Chicago"}

};

foreach (DictionaryEntry row in cities)

{

  Console.WriteLine(row.Key + " " + row.Value);

}

// See if the Hashtable contains this key.

bool b = hashtable.ContainsKey("cat");

// Test the Contains method.

Console.WriteLine(hashtable.Contains("dog"));

// Get value of cat key.

int v = (int)hashtable["bird"];

Console.WriteLine(v);

**Tuple**

**Tuple در دات نت فریم ورک 4 معرفی شد. تاپل ها مجموعه ای المنت‌ها از نوع‌های (data type) مخلتف هستند.**

Tuple<T1, T2, T3, T4, T5, T6, T7, TRest>

**تعریف Tuple:**

// 1

Tuple<int, string, bool> tuple = new Tuple<int, string, bool>(1,"Bill", true);

//2

var person = Tuple.Create(1,"steve","jobs", true, "apple", "company", 4, 50);

**نکته: Tupleها حداکثر تا 8 المنت را در خود ذخیره می‌کنند. در صورتی بیشتر از 8 المنت در یک Tuple قرار دهید، compile-error می‌دهد.**

**نکته: تاپل‌ها تغییر ناپذیر هستند (immutable) از این بعد از تعریف یک تاپل نمی‌توانیم آن را تغییر دهیم.**

**دستیابی به المنت‌های یک Tuple**

**برای دستیابی به المنت‌های یک Tuple از Item<elementNumber> استفاده می‌کنیم. برای مثال: item1، item2، item2 الی item7. به این شکل که item1 اولین المنت را برمی‌گرداند. (return می‌کند). Item2 دومین المنت را بر‌می‌گرداند و به همین ترتیب تا item7. آخرین المنت که می‌شود المنت 8 توسط Rest برگردانده می‌شود.**

var numbers = Tuple.Create("One", 2, 3, "Four", 5, "Six", 7, 8);

numbers.Item1; // returns "One"

numbers.Item2; // returns 2

numbers.Item3; // returns 3

numbers.Item4; // returns "Four"

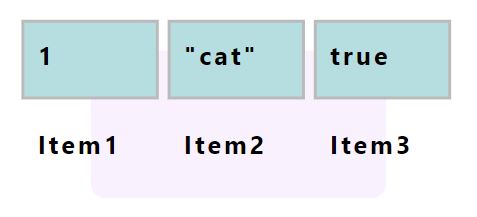
numbers.Item5; // returns 5

numbers.Item6; // returns "Six"

numbers.Item7; // returns 7

numbers.Rest; // returns (8)

numbers.Rest.Item1; // returns 8

****

**Tupleهای تو در تو (Nested Tuple)**

**اگر می‌خواهید بیشتر از 8 المنت در یک تاپل قرار دهید، باید به شکل تو در تو استفاده کنید. مانند زیر:**

var numbers = Tuple.Create(1, 2, 3, 4, 5, 6, 7, Tuple.Create(8, 9, 10, 11, 12, 13));

numbers.Item1; // returns 1

numbers.Item7; // returns 7

numbers.Rest.Item1; //returns (8, 9, 10, 11, 12, 13)

numbers.Rest.Item1.Item1; //returns 8

numbers.Rest.Item1.Item2; //returns 9