INITIALIZING A DICTIONARY:

* Dictionary<keys, values> dict = new Dictionary<keys, values>(); or
* var dict = new Dictionary<string, int>(); or
* var dict = new SortedDictionary<string, int>(); // - sorts the keys in the dictionary;

Example:

var phonebook = new Dictionary<string, string>();

phonebook[“John Smith”] = “+ 359 888 888 888”; // - in this case we add and replace, unlike the Add() operation which doesn’t replace the item;

phonebook[“Adriana Lima”] = “+ 359 777 777 777”;

LOOPING THROUGH THE DICTIONARY:

foreach (KeyValuesPair<string, string> pair in phonebook) // - KeyValuesPair<string, string> is a substitute to var;

{

Console.WriteLine(“{0} -- > {1}”, pair.Key, pair.Value);

}

COMBINING DICTIONARIES WITH LISTS:

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

marks[“Ivan”] = new List<int>{1, 2, 3, 4};

marks[“Ivan”].Add(6); // - adding one more mark to Ivan;

BASIC OPERATIONS:

* Add() – adds item in the dictionary; //- with Add() we don’t replace if the item already exists;

Example: phonebook.Add(“Candice Swanepoel”, 3);

* Remove() – removes item from the dictionary;

Example: phonebook.Remove(“John Smith”);

* Clear() – clears the entire dictionary;

Example: phonebook.Clear();

OTHER OPERATIONS:

* Count – counts the items in the dictionary;

Example: Console.WriteLine(phonebook.count);

* Keys – shows all the keys in the dictionary;

Example: phonebook.Keys;

* Values – shows all the values in the dictionary;

Example: phonebook.Values;

* TryParse() – with this method, we prevent the program from failing if we don’t use the correct (string / int and so on) format;

Example:

int number = 0;

string text = Console.Readline();

bool parsed = int.TryParse(text, out number); // - “out” means that we are going to write the variable in number (already declared) if valid;

* Replace(“ “, string.Empty); - removing spaces between words in a string;

Example:

var text = “Ivan went to school and he got a 2!”

.ToLower().

.Replace(“ “, string.Empty);

* Reverse() – reversing the elements in the collection;

Example:

int[] nums = { 1, 2, 3, 4, 5, 6};

nums = nums.Reverse(); // nums = 6, 5, 4, 3, 2, 1

* Concat() – concatenates elements from two collections

Example:

int[] nums = { 1, 2, 3, 4, 5, 6 };

int[] otherNums = { 7, 8, 9, 0 };

nums = nums.Concat(otherNums); // nums = 1, 2, 3, 4, 5, 6, 7, 8, 9, 0

BOOLEAN METHODS:

* ContainsKey() – checks if a key is present in the dictionary;
* ContainsValue() – checks if a value is present in the dictionary;

Example:

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

dictionary[“Pesho”] = 6;

Console.WriteLine(dictionary.ContainsKey(“Pesho”)); // - will return true;

Console.WriteLine(dictionary.ContainsValue(6)); // - will return true;

* TryGetValue() – check if a key is present in the dictionary and outputs the value, or returns the default value of the type;

Example:

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

dictionary[“Pesho”] = 6;

int number = 0;

bool valueExists = dictionary.TryParse(“Pesho”, out number);

Console.WriteLine(valueExists); // - will return true;

Console.WriteLine(number); // - will return 6;

MATH OPERATIONS:

* Min() – finds the smallest element in a collection;

Example:

var list = new list<int> {2, 3, 4, 5, 6};

var min = list.Min();

* Max() – finds the largest element in a collection;

Example: list.Max();

* Sum() – finds the sum of all elements in a collection;

Example: list.Sum();

* Average() – finds the average of all elements in a collection;

Example: list.Average();

Hash Function (definition) – a function thanks to which a given dictionary makes difference between the different items in it;

READING COLLECTIONS ON A SINGLE LINE:

var nums = Console.ReadLine()

.Split()

.Select((number) => double.Parse(number)); // - a substitute for

.Select(double.Parse);

CONVERING COLLECTIONS:

var nums = Console.ReadLine()

* ToArray() – converting to array;
* ToList() – converting to list;
* ToDictionary(key, value) – converting to dictionary;
* ToCharArray() – converting to array;

SORTING COLLECTIONS:

* OrderBy() – sort collections;

Example:

var list = new List<int>{1, 2, 3, 4, 5};

var result = list

.OrderBy(number => number)

.ToList();

foreach (var item in result)

{

Console.WriteLine(item);

}

* OrderByDescending(number => number) // - sorts the result in descending order;
* ThenBy() – sorts collections by more than one criteria;

Example:

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

Dictionary<int, string> sortedDict = products

.OrderBy(pair => pair.Value)

.ThenBy(pair => pair.Key)

.ToDictionary(pair => pair.Key, pair => pair.Value);

* ThenByDescending() – sorting in descending order;

TAKE / SKIP N ELEMENTS FROM COLLECTION:

* Take() – returns elements from the dictionary;
* Skip() – skips elements from the dictionary;

Example:

var list = new List<int>{1, 2, 3, 4, 5};

var result = list

.Skip(2) // - if we combine Skip and Take, it will return 3, 4 and 5;

.Take(3) // - returns the first three elements … in this case 1, 2 and 3;

.ToList();

LAMBDA EXPRESSIONS:

* A lambda expression is an anonymous function containing expressions and statements
* Lambda expressions: Use the lambda operator **=> ……** Read as "goes to". The left side specifies the input parameters. The right side holds the expression or statement.
* Lambda functions are inline methods (functions) that take input parameters and return values:

x => x / 2 ⬄ static int Func(int x) { return x / 2; };

x => x != 0 ⬄ static bool Func(int x) { return x != 0; };

() => 42 ⬄ static int Func() { return 42; };

FILTER COLLECTIONS:

* Where() – allows us to find all elements that respond to a specific condition;
* Count() – find the count of those elements that respond to a specific condition;
* Any() – shows if there is at least one element we look for in the dictionary;
* All() – all of the elements should be present;
* Select() – transforms the current collection into something else;
* Distinct() – shows only the unique elements in the collection;
* Split() – splits the elements in a specific way;

Example:

var list = new List<int> { 2, 4, 3, 12, 11, 10, 7, 2, 2, 2, 3, 3, 3};

var result = list // - here we create new collection

.Where(n => n % 2 == 0)

.Count(n => n > 5 && n < 10 )

.Any(n => n > 2) // - returns true of false

.All(n => n > 200) // - returns true of false…in this case false

.Select(n => 2 \* n) // - multiplies all elements by 2

.Select(n => (char)(n + ‘a’ - 1)); - shows the letters in alphabet, corresponding to the numbers in the list

.Distinct() // - shows all unique elements without any repetition

.Split(new [] {‘ ‘, ‘, ‘, ‘. ‘ }, StringSplitOptions.RemoveEmptyEntries)

.ToArray();

foreach (var item in result)

{

Console.WriteLine(item); // - will return all even elements;

}

TAKE SINGLE ELEMENTS FROM COLLECTION:

* First() – returns the first element that responds to specific requirement;
* FirstOrDefault() – if no element found, it returns the default value of the element;
* Last() – returns the last element that responds to specific requirement;
* LastorDefault() – if no element found, it returns the default value of the element;
* Single() – returns the single element the responds to specific requirement;
* SingleorDefault() - if no element found, it returns the default value of the element;

Example:

var list = new List<int>{ 1, 2, 3, 4, 5, 6};

var first = list.First() // - returns the first element which is 1;

var first = list.First(n => n % 2 == 0) // - returns the first even element…in this case 2;

var last = list.Last(n => n % 2 == 0) // - returns the last even element…in this case 6;

var single = list.Single(n => n % 2 == 0) // - in this case there will be an error message because there are more than one even element;