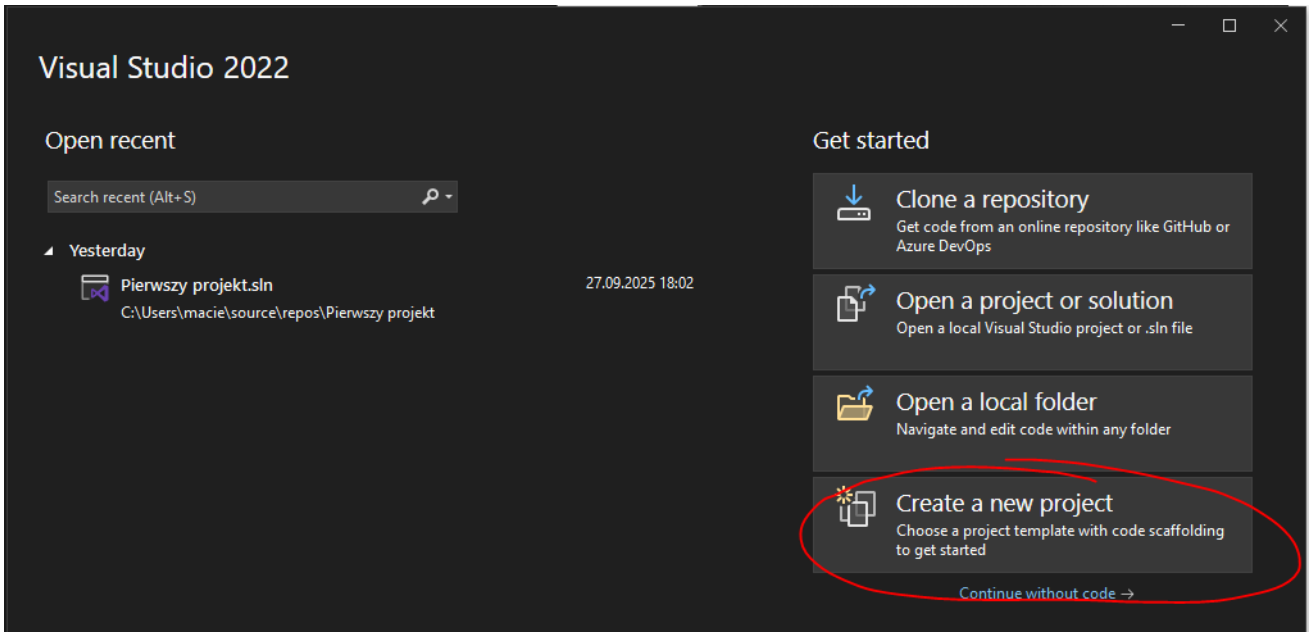
Programowanie równoległe Lab 1

Wprowadzenie

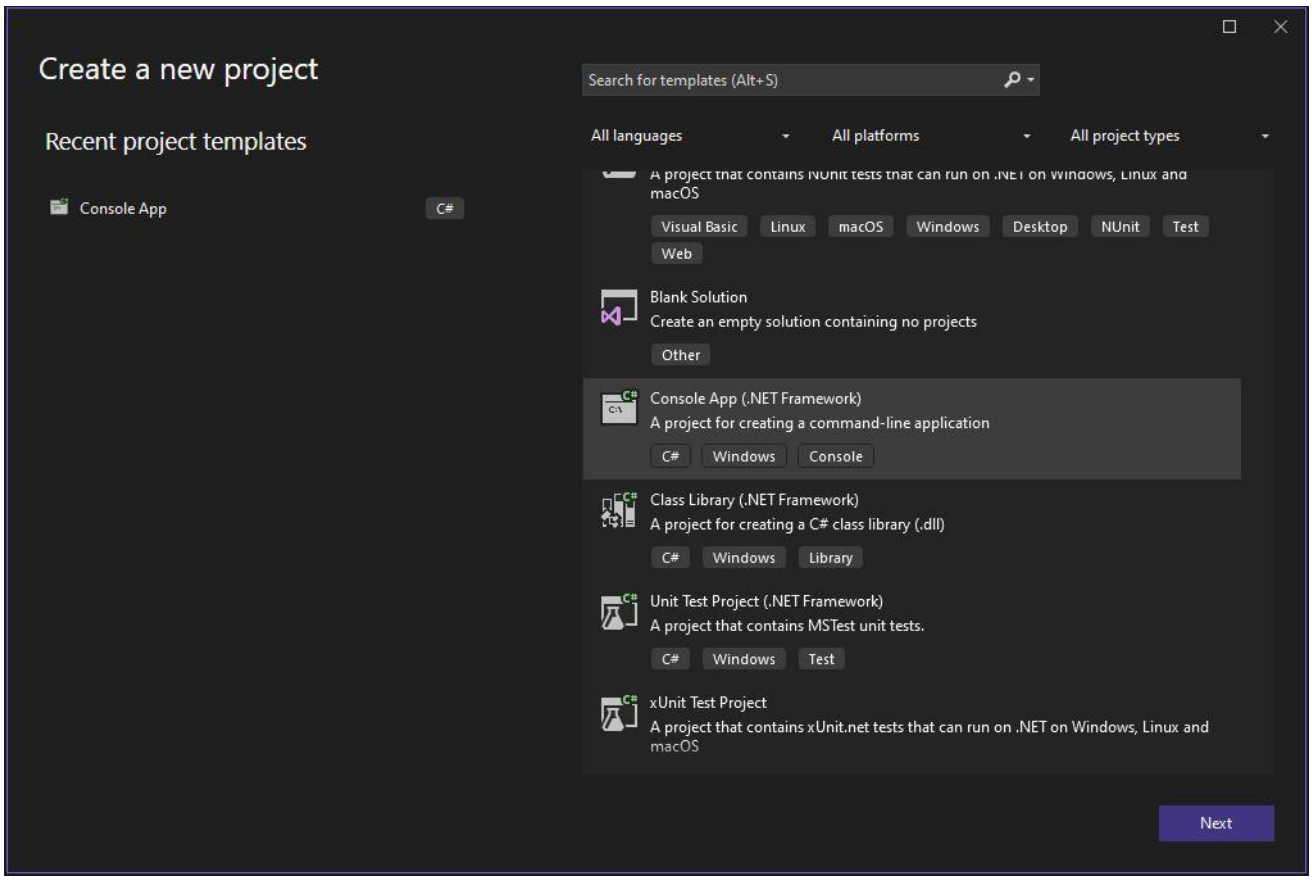
Przedstawione wprowadzenie zostało zrobione na wersji 2022 Visual Studio.

Poprzednie wersje Visual Studio mogą się nieco różnić, również mogą być niedostępne niektóre konstrukcje języka programowania.

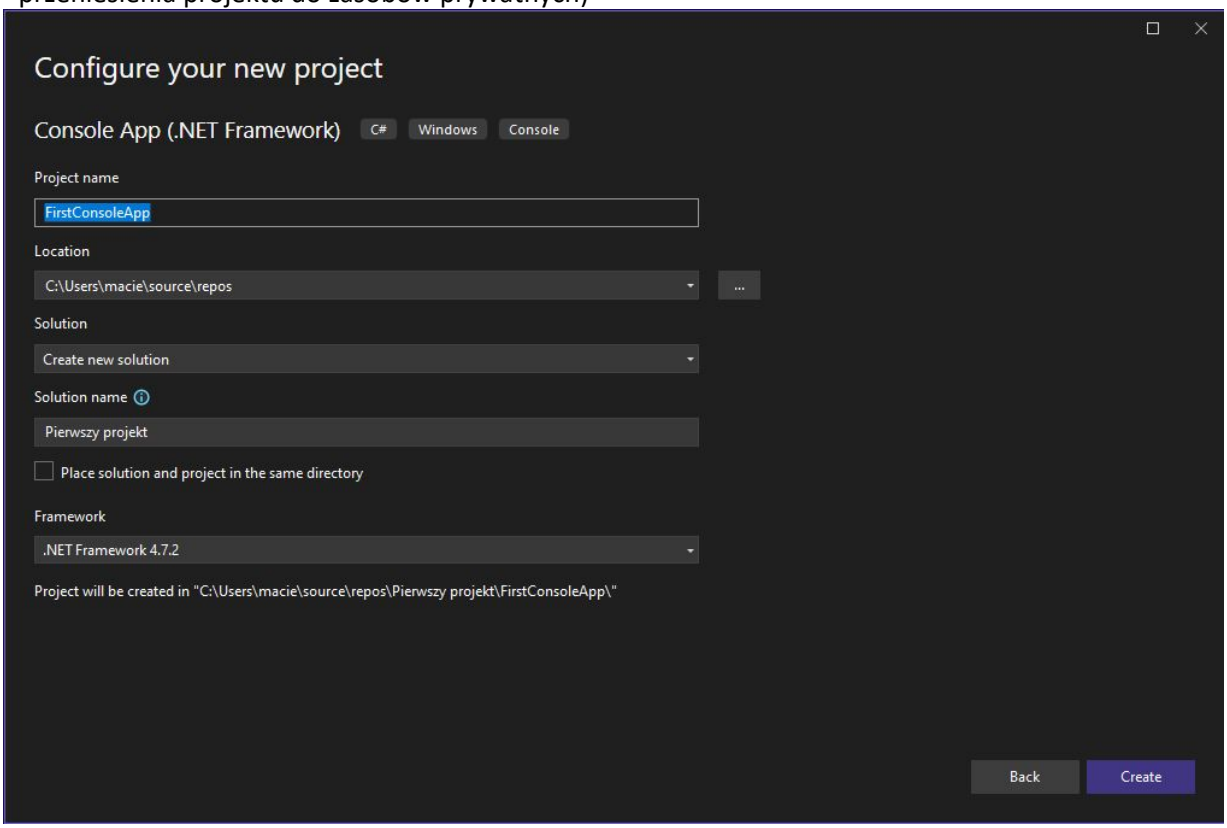
1. Uruchamiamy Visual Studio i w oknie wyboru projektu klikamy Create New Project



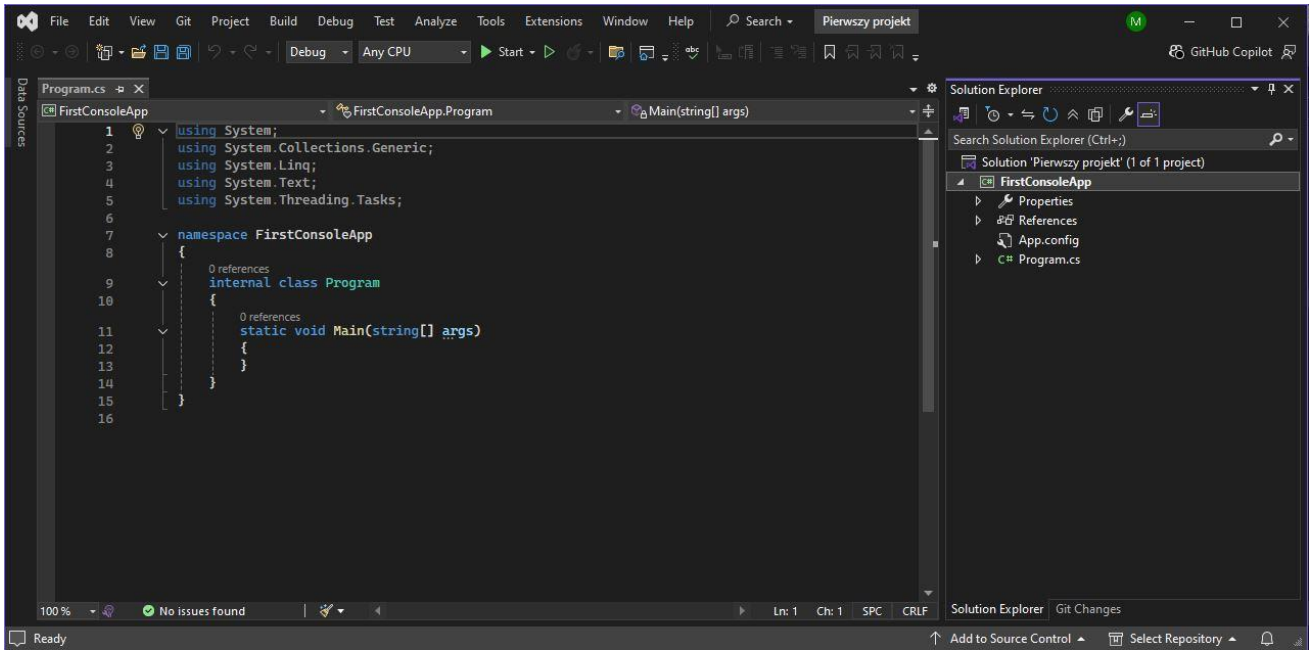
2. Wyszukujemy projekt typu C# Console App (.NET Framework)
(wybieramy wersję C# a nie VB i wersję Framework a nie Console App (.NET Core))



3. Wpisujemy nazwę, zapamiętujemy lokalizację (lokalizację można zmienić celem późniejszego przeniesienia projektu do zasobów prywatnych)

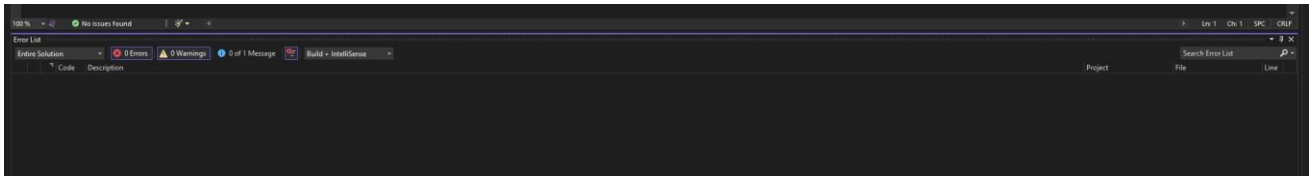


4. Najważniejsze elementy okna



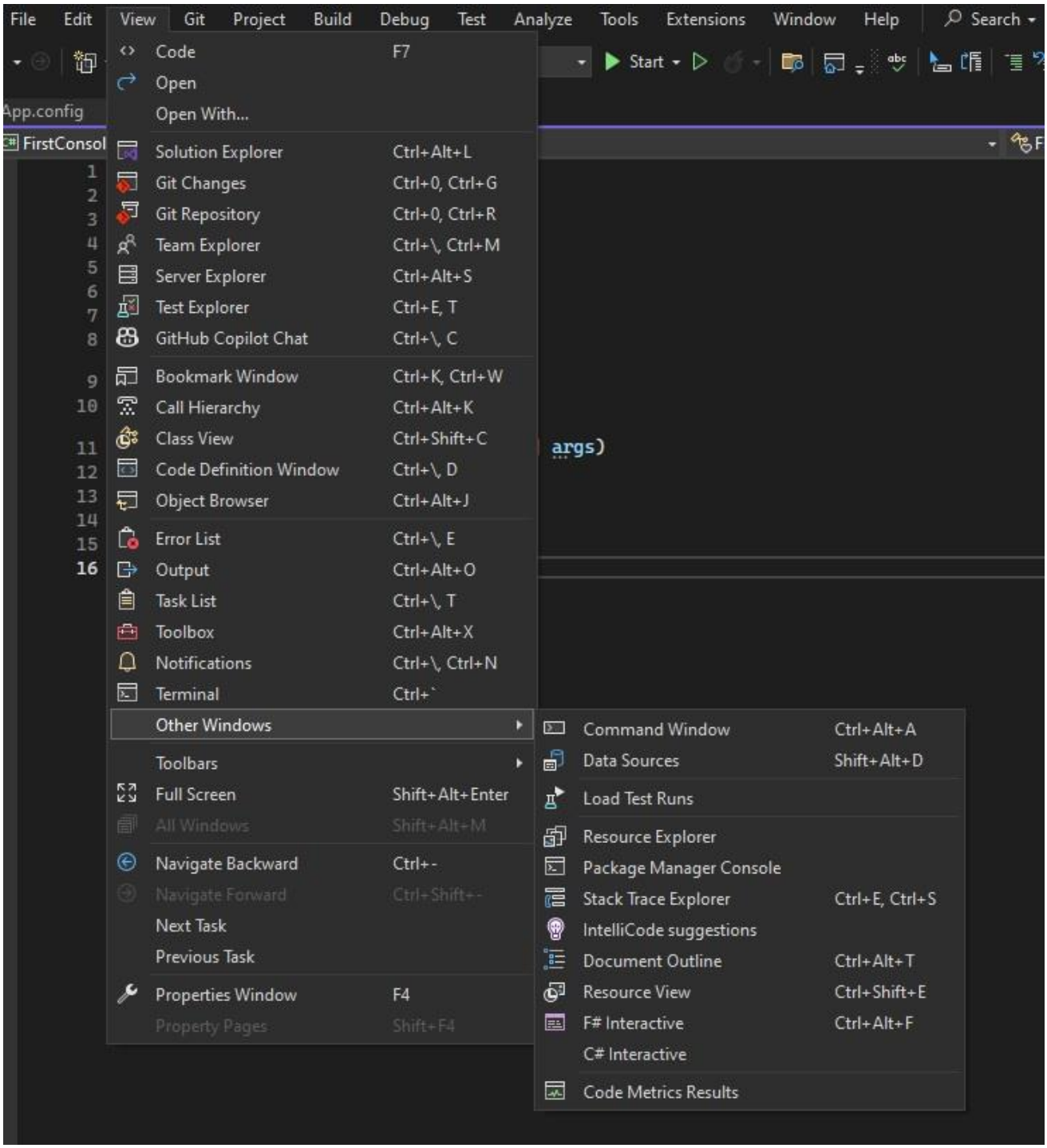
Solution Explorer – wyświetla solucję wraz ze wszystkimi elementami. Solucja zawiera projekt (PierwszyProjekt), projekt zawiera między innymi pliki App.config (konfiguracyjny), Program.cs (klasa Program)

Error List – informuje nas o błędach w kodzie (warto czytać)

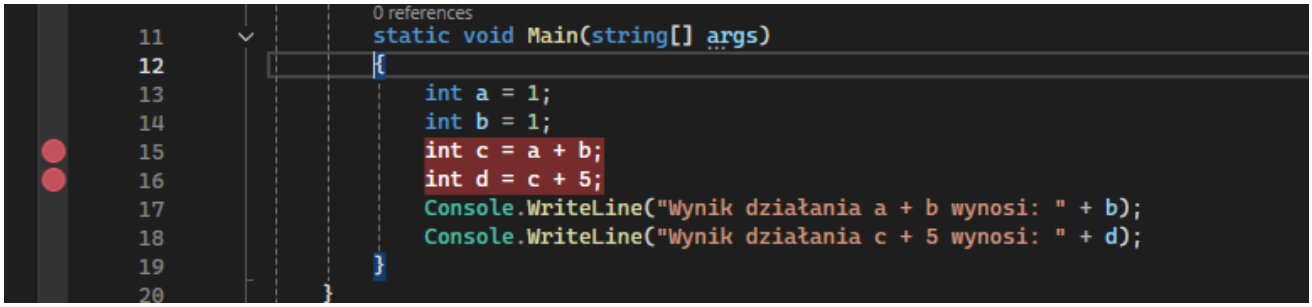


Program.cs – otwarty plik Program.cs. Program.cs na początku zawiera dyrektywy using odpowiedzialne za skracanie odwołań do zewnętrznych obiektów, dalej zdefiniowano przestrzeń nazw FirstConsoleApp, która domyślnie jest zgodna z nazwą projektu i może być zmieniona. Przestrzeń nazw służy do logicznego grupowania komponentów w aplikacji. W przestrzeni nazw zdefiniowano klasę Program. Klasa Program zawiera tylko jedną metodę Main. Metoda Main jest miejscem z którego program zaczyna swoje działanie.

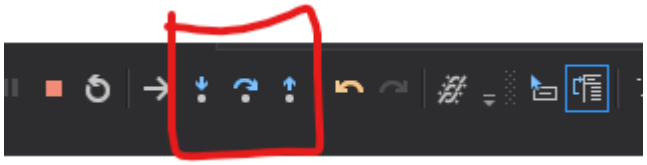
W przypadku braku któregoś ze wskazanych okien można je otworzyć z menu View. Znajdują się tam ww. pozycje oraz wiele innych.

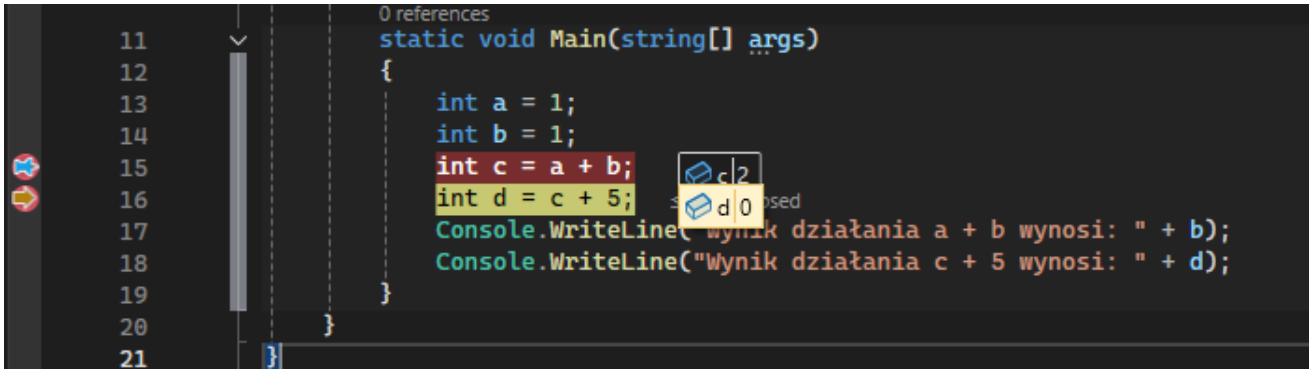


Na lewej listwie okna głównego możemy ustawić tzw. Pułapkę (breakpoint), celem debugowania kodu

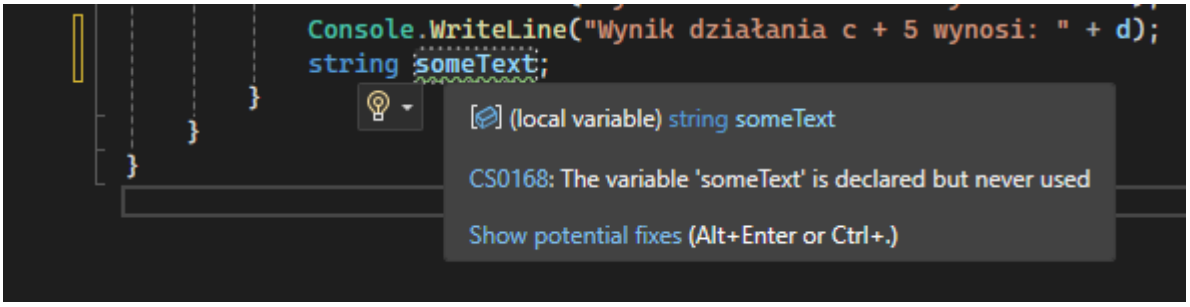


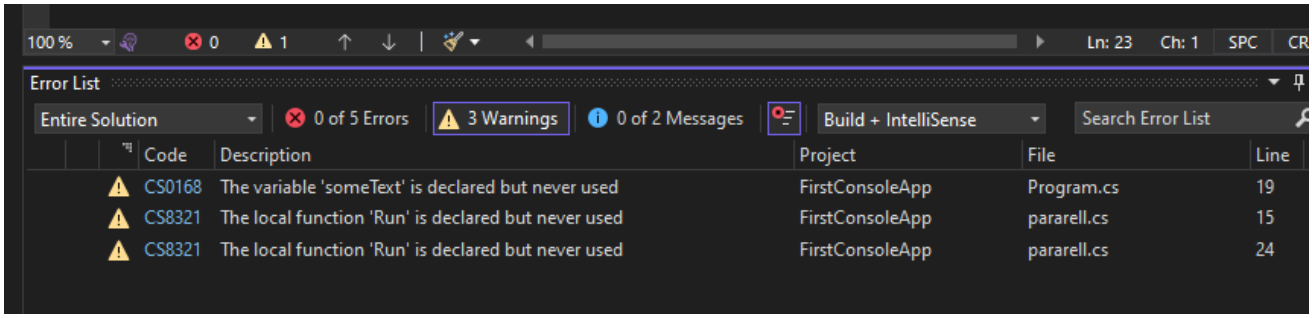
Po uruchomieniu program w tym miejscu przerwie swoje działanie, a my będziemy mogli kontynuować je (menu lub F10, F11) krok po kroku kontrolując stan aplikacji.





5. Lista błędów podpowiada często bardzo precyzyjnie w czym jest problem, ikona żaróweczki sugeruje rozwiązanie problemu. Sugerowane rozwiązanie trzeba stosować z rozmysłem, ponieważ często są podawane dwie lub trzy propozycje rozwiązania, ale właściwe rozwiązanie zależy od kontekstu problemu.





6. Klasy w C# są miejscem, w którym możemy tworzyć swój kod.

W poniższym przykładzie w klasie Program utworzono statyczną metodę Dodaj przyjmującą dwa parametry i zwracającą wynik dodawania.

Metodę dodaj wywołano w metodzie Main, a wynik zwrócony przez metodę Dodaj został przypisany do zmiennej lokalnej wynik.

Pamiętajmy że metoda Main stanowi punkt startowy aplikacji, jest uruchamiana przez środowisko podczas uruchamiania aplikacji.

```
class Program
{
    0 references
    static void Main(string[] args)
    {
        int wynik = Dodaj(1, 2);
    }

    1 reference
    static int Dodaj(int wartosc1, int wartosc2)
    {
        return wartosc1 + wartosc2;
    }
}
```

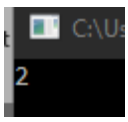
7. Do wywołania metody Dodaj możemy wykorzystać zmienne. W przykładzie poniżej wykorzystano statyczną zmienną prywatną a deklarowaną na poziomie klasy oraz prywatną zmienną b deklarowaną na poziomie metody.

Miejsce deklaracji zmiennych oraz modyfikatory są istotne ze względu na możliwość bezpośredniego dostępu do zmiennej.

```
class Program
{
    private static int a = 1;
    0 references
    static void Main(string[] args)
    {
        int b = 2;
        int wynik = Dodaj(a, b);
    }
```

8. Do wyświetlenia wybranej wartości na ekranie możemy wykorzystać klasę Console.

```
static void Main(string[] args)
{
    int b = 2;
    Console.WriteLine(b);
}
```



9. Aktualnie program kończy swoje działanie bezpośrednio po wykonaniu wszystkich instrukcji. Aby wstrzymać działanie możemy wykorzystać metodę ReadKey() z klasy Console. Metoda ta wstrzymuje działanie programu i czeka na wciśnięcie klawisza przez operatora.

```
static void Main(string[] args)
{
    int b = 2;
    Console.WriteLine(b);

    Console.ReadKey();
}
```

10. Za pomocą metody ReadLine() z klasy Console możemy odczytać tekst wprowadzony przez użytkownika.

```
static void Main(string[] args)
{
    Console.WriteLine("Podaj jakiś tekst");
    string txt = Console.ReadLine();

    Console.WriteLine("Wpisano tekst: {0}", txt);
    Console.ReadKey();
}
```

```
Podaj jakiś tekst
Podaję dziwny tekst
Wpisano tekst: Podaję dziwny tekst
```

11. Wpisany tekst można konwertować na liczbę metodami podanymi w wykładzie.

```
static void Main(string[] args)
{
    Console.WriteLine("Podaj liczbę");
    string txt = Console.ReadLine();

    int liczba = int.Parse(txt);
    Console.WriteLine("Wpisano liczbę: {0}", txt);
    Console.ReadKey();
}
```

```
Podaj liczbę
10
Wpisano liczbę: 10
```

13. Bądź gotowy na to że użytkownik wpisze zamiast liczby tekst.

```
Podaj liczbę
abc
```

```
    Console.WriteLine("Podaj liczbę");
    string txt = Console.ReadLine();

    int liczba = int.Parse(txt);
    Console.WriteLine("Wpisano licz e: {0}", txt);
    Console.ReadKey();
}


0 references
static int Dodaj(int wartosc1, in
{
    return wartosc1 + wartosc2;
}
```

Exception Unhandled

System.FormatException: 'Input string was not in a correct format.'

This exception was originally thrown at this call stack:
System.Number.StringToNumber(string, System.Globalization.
System.Number.ParseInt32(string, System.Globalization.Numb
int.Parse(string)

O wyjątkach będziemy mówić na przyszłych wykładach, tymczasem dopilnuj aby wyjątek nie pojawił się w Twoim programie.

13. Za pomocą metody WriteLine możemy wyświetlić ciąg tekstu

```
static void Main(string[] args)
{
    int b = 2;
    Console.WriteLine(b);

    Console.WriteLine("Naciśnij dowolny klawisz ...");
    Console.ReadKey();
}
```

```
C:\Users\konrad\source\repos\PierwszyProjekt\
2
Naciśnij dowolny klawisz ...
```

14. Możemy również łączyć tekst ze zmiennymi. Łączenie możemy robić na kilka omówionych w wykładzie sposobów.

```
static void Main(string[] args)
{
    int b = 2;
    Console.WriteLine("Wartość b jest równa " + b);
    b = b + 1;
    Console.WriteLine("Wartość b jest równa {0}", b);
    Console.WriteLine($"Wartość b jest równa {++b}");
    b += 1;
    Console.WriteLine("Wartość b jest równa {0:F2}", b);

    Console.WriteLine();
    Console.WriteLine("Naciśnij dowolny klawisz ...");
    Console.ReadKey();
}
```

```
Wartość b jest równa 2
Wartość b jest równa 3
Wartość b jest równa 4
Wartość b jest równa 5,00

Naciśnij dowolny klawisz ...
```

15. Możemy również wprowadzić warunki umożliwiające sterowanie programem

```
static void Main(string[] args)
{
    Console.WriteLine("Podaj liczbę");
    string txt = Console.ReadLine();

    int liczba = int.Parse(txt);

    if(liczba>3)
    {
        Console.WriteLine("Wpisano liczbę: {0}", txt);
    }
    else
    {
        Console.WriteLine("Coś wpisano, ale nie chcesz wiedzieć co.");
    }
    Console.ReadKey();
}
```

```
Podaj liczbę
17
Wpisano liczbę: 17
```

```
Podaj liczbę
2
Coś wpisano, ale nie chcesz wiedzieć co.
```

Więcej sposobów na utworzenie warunku znajdziesz w wykładzie.

16. Kilka przykładów deklaracji zmiennych, w tym również tablic.

```
class Program
{
    private static int a = 1;
    0 references
    static void Main(string[] args)
    {
        int b = 2;
        string txt = "jakiś tekst";
        double liczba1 = 1.34;

        int[] c = new int[5];
        int[] d = new int[] { 4, 6, 8, 99 };


        Console.WriteLine("Naciśnij dowolny klawisz ...");
        Console.ReadKey();
    }
```

17. Użycie pętli for do wyświetlenia zawartości tablicy

```
static void Main(string[] args)
{
    int[] d = new int[] { 4, 6, 8, 99 };

    for (int i = 0; i <= d.Length - 1; i++)
    {
        Console.WriteLine("Element {0} ma wartość {1}", i, d[i]);
    }

    Console.WriteLine();
    Console.WriteLine("Naciśnij dowolny klawisz ...");
    Console.ReadKey();
}
```

```
Element 0 ma wartość 4
Element 1 ma wartość 6
Element 2 ma wartość 8
Element 3 ma wartość 99

Naciśnij dowolny klawisz ...
```

18. Użycie pętli for do zmiany wartości elementów w tablicy (podniesienia do kwadratu) oraz pętli foreach do wyświetlenia wartości elementów po zmianie.

```
static void Main(string[] args)
{
    int[] d = new int[] { 4, 6, 8, 99 };

    for (int i = 0; i <= d.Length - 1; i++)
    {
        d[i] = d[i] * d[i];
    }

    foreach (int w in d)
    {
        Console.WriteLine("Element ma wartość {0}", w);
    }

    Console.WriteLine();
    Console.WriteLine("Naciśnij dowolny klawisz ...");
    Console.ReadKey();
}
```

```
Element ma wartość 16
Element ma wartość 36
Element ma wartość 64
Element ma wartość 9801

Naciśnij dowolny klawisz ...
```

Klasy

19. Klasy stanowią element niezbędny, w programowaniu obiektowym, do tworzenia modułowego i spójnego kodu aplikacji. Zgodnie z zasadą pojedynczej odpowiedzialności klasa powinna odpowiadać za jedną rzecz np. dokonanie obliczeń, a jeśli potrzebujemy wykonać inną rzecz np. zapis do bazy przeliczonych danych, powinna ona być realizowany przez inną klasę odpowiedzialną za zapis danych.

20. Definicje klasy z zasady powinny być umieszczane w oddzielnych plikach. Przykładowe klasy zaprezentowano poniżej.

21. Klasa Function2X ma właściwość (atrybut) Name zwracającą przyjazną dla użytkownika nazwę funkcji oraz metodę GetY, zwracającą wartość funkcji dla podanego argumentu x.

```
internal class Function2X
{
    0 references
    public string Name { get; set; } = "y=2x";

    0 references
    public double GetY(double y)
    {
        return 2 * y;
    }
}
```

22. Klasa Calculator ma właściwość Name zwracającą nazwę kalkulatora, oraz metodę GetValue dokonującą obliczeń z wykorzystaniem klasy Function2x.

```
internal class Calculator
{
    0 references
    public string Name { get; set; } = "Main calculator";

    0 references
    public double GetValue(Function2X func, double from, double to)
    {
        double result = 0;

        while (from <= to)
        {
            result += func.GetY(from);
            from += 0.01;
        }
        return result;
    }
}
```

Jak możemy zaobserwować instancję klasy Function2x należy przekazać do metody GetValue.

Metoda GetValue uruchomi metodę GetY i pobierze wyniki zgodnie ze swoimi potrzebami.

23. Całość uruchamiamy zgodnie z przykładem poniżej

```
static void Main(string[] args)
{
    Calculator calculator = new Calculator();
    Function2X function2X = new Function2X();

    Console.WriteLine($"{calculator.Name} processed function {function2X.Name} " +
        $" and got result {calculator.GetValue(function2X, 1, 2)} ");

    Console.ReadKey();
}
```

A po uruchomieniu rezultat zostaje wyświetlony w oknie konsoli.

```
Main calculator processed function y=2x and got result 299
```

Co zrobić gdy potrzebujemy wykonać te same obliczenia ale dla różnych funkcji?

Odpowiedź ukryta jest w interfejsach.

Interfejsy

24. Interfejs jest to „prototyp” klasy. Definiuje on tylko publiczne właściwości i metody ale bez kodu stanowiącego logikę metody.

```
internal interface IFunction
{
    4 references
    string Name { get; }
    3 references
    double GetY(double x);
}
```

25. Interfejs możemy użyć jako parametr przekazywany do metody i zwracany z metody

W tym przypadku do metody GetValue przekazujemy obiekt typu IFunction, podobnie jak wcześniej przekazywaliśmy obiekt typu Function2x

```
internal interface ICalculator
{
    3 references
    string Name { get; }
    3 references
    double GetValue(IFunction function, Range range, Action<int> progress);
    5 references
    int Steps { get; }
}
```

26. W powyższym przykładzie brakuje nam obiektu Range. Zaimplementuj klasę Range zgodnie z poniższym przykładem.

```
internal class Range
{
    1 reference
    public decimal From { get; set; }
    1 reference
    public decimal To { get; set; }
    1 reference
    public decimal Step { get; set; }

    0 references
    public Range(decimal from, decimal to, decimal step)
    {
        From = from;
        To = to;
        Step = step;
    }
}
```

27. Kilka klas może implementować jeden interfejs. Wszystkie one stanowią obiekty typu „Interfejs”, ale różnią się logiką. Poniżej dwa przykłady

```
internal class Function2X : IFunction
{
    3 references
    public string Name => "2x";

    2 references
    public double GetY(double x)
    {
        return 2 * x;
    }
}
```

```
internal class FunctionXhalf : IFunction
{
    3 references
    public string Name => "x/2";

    2 references
    public double GetY(double x)
    {
        return x/2;
    }
}
```

28. Obiekty typu „Interfejs” (a takiego typu są obie powyższe klasy) możemy przekazywać do metod, który konkretnie obiekt zostanie przekazany decyduje logika aplikacji podczas działania programu (np. użytkownik ma możliwość wyboru) a nie programista podczas tworzenia kodu.

```
1 reference
public void Run(int iterationsNo)
{
    ICalculator calculator = new IntegralCalculator();
    IFunction function = new FunctionXhalf();
    Range range = new Range(0, 2, 1);

    for (int i = 1; i <= iterationsNo; i++)
    {
        var result = calculator.GetValue(function, range, (value) => ProgressChanged(range.Step.ToString(), value) );
        Console.WriteLine($"{calculator.Name} result for {function.Name} with step {range.Step} from 0 to 2 is: {result}        total steps:{calculator.Steps}");
        Completed(range.Step.ToString());
        range.Step /= 10;
    }

    Console.WriteLine("\n\nPress key ....");
    Console.ReadKey();
}
```

Innymi słowy tym sposobem możemy zmieniać funkcjonalność kodu (podmieniając zdefiniowane Instancje typu IFunction, bez konieczności zmiany sposobu liczenia (Calculator)

Powyższy przykład nie jest w pełni zaimplementowany. W ramach doskonalenia spróbuj zaprogramować wszystkie niezbędne komponenty tak aby powyższy kod działał.

Zadanie 1.

Uzbrojony w powyższą wiedzę zrób program, który policzy wyrazy ciągu Fibonacciego.

Programistycznie możemy powiedzieć, że ciąg Fibonacciego jest to tablica, w której:

- Wartość elementu zerowego jest równa zero
- Wartość elementu pierwszego jest równa jeden
- Wartość każdego następnego elementu ciągu Fibonacciego jest sumą dwóch poprzednich elementów ciągu.

Kilka pierwszych elementów przedstawiono w poniższej tabeli:

0	1	1	2	3	5	8	13	21	34

Program powinien zapytać użytkownika o ilość elementów ciągu, następnie powinien wyliczyć i wyświetlić elementy ciągu.

UWAGA!!!

Każde zadanie stwórz w oddzielnej klasie, w metodzie Main rób tylko instancję klasy oraz wywołanie stworzonej metody.

Zadanie 2.

Powyższe zadanie przerób w taki sposób, aby program pytał użytkownika o dwie liczby, następnie wyświetlił tyle elementów, ile podał użytkownik w drugiej liczbie, zaczynając od elementu, który był podany pierwszą liczbą. Zadbaj o właściwe informacje/komunikaty dla użytkownika.

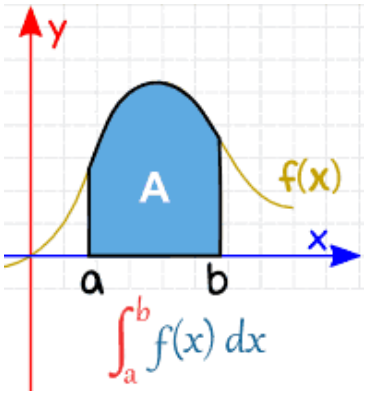
Przykład: L1=5, L2=3, wynik: 5, 8, 13 https://mathdf.com/int/pl/

Zadanie 3.

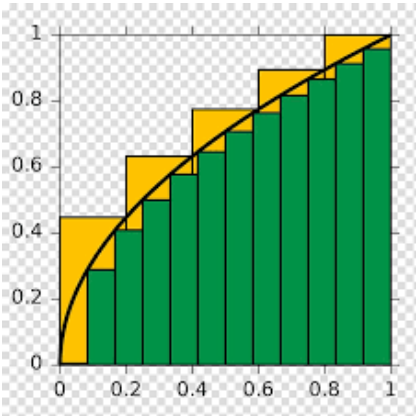
Numeryczne liczenie całki.

O całce możemy myśleć jak o polu powierzchni pomiędzy wykresem funkcji a osią x.

Kalkulator online: https://mathdf.com/int/pl/



Numeryczne liczenie całki możemy wykonać poprzez podział przedziału na bardzo małe odcinki, następnie liczeniu dla każdego odcinka wartości funkcji, po czym aproksymacji wartość całki dla odcinka za pomocą pola prostokąta lub trapezu. Suma wszystkich policzonych pól stanowi numerycznie wyliczoną wartość całki.



Zadanie polega na numerycznym wyznaczeniu wartości całki dla funkcji y=1/2*x na przedziale x od 0 do 2 z wykorzystaniem prostokątów jako elementów aproksymujących.

Zadanie 4.

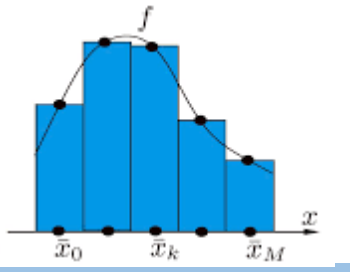
Zbadaj wpływ zmiany ilości elementów przedziału, dla którego wyznaczamy całkę (szerokości elementu) na dokładność wyznaczania wartości całki.

Zadanie 5.

Zbadaj wpływ zmiany punktu wyliczania wartości funkcji na dokładność wyznaczania całki.

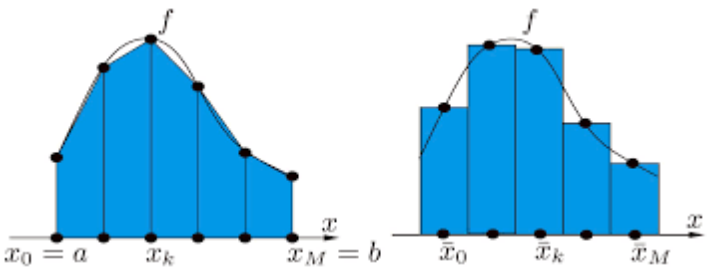
Wartość funkcji może być wyznaczana np. po lewej stronie prostokąta, stronie prawej lub na środku.

Rysunek reprezentuje ostatnią z wyżej wymienionych możliwość.



Zadanie 6.

Zbadaj wpływ zmiany elementu aproksymującego z prostokąta na trapez na dokładność wyznaczania całki. Poniższy rysunek pokazuje zasadę wykorzystania trapezów i prostokątów.



Zadanie 7.

Wykonaj program do wyznaczania wartości całki dla funkcji y=sin(x) na przedziale x od 0 do 2π.

Zadanie 8.

Wykonaj program do wyznaczania wartości całki dla funkcji y=ax2+bx +c. Przy czym parametry a, b i c powinny zostać podane przez użytkownika.

Podsumowanie i przesłanie wyników ćwiczeń:

1. Utwórz plik ZIP nazwany **wyniki_Lab_1_2_Nazwisko_Imie_*yyyy_mm_dd.zip***
2. W pliku umieść rezultat twojej pracy ze wszystkich zadań.
3. Gotowy plik ZIP umieść na platformie e-learningowej.

Ocenianie:

1. Prace oddane w terminie – ocean z zakresu 5 - 2.
2. Prace oddane do 7 dni po terminie - ocean z zakresu 4.5 - 2.
3. Prace oddane 7-14 dni po terminie - ocean z zakresu 4 - 2.
4. Prace oddane 14-30 dni po terminie - ocean z zakresu 3 - 2.

Skalowanie ocen:

3 (dst.) –	51-60%
3+ (dst. plus) –	61-70%
4 (db.) -	71-80%
4+ (db. plus) -	81-90%
5 (bdb.) -	91-100%