Hello World!

A więc otwieramy Visual Studio > Plik > Nowy > Projekt > Visual F# > **Aplikacja konsolowa**.

W pliku Program.fs usuwamy wszystko i wpisujemy:

```
open System
printfn "Hello World"
Console.ReadKey() |> ignore
```

I klikamy Start lub Ctrl+F5.

Przeanalizujmy wprowadzony kod:

`open System` – udostępnienie przestrzeni nazw (biblioteki) – pozwala nam uniknąć pisania wieloczłonowych nazw, np. `System.Console.WriteLine("Hello World")`

`printfn "Hello World"` - wydruk podanego łańcucha tekstowego przejście do następnego wiersza (warto zaznaczyć, że `printf` wykonuje się bez zmiany wiersza)

`Console.WriteLine("Hello World")` - inny sposób na wyprowadzenie wyników i komunikatów

`Console.ReadKey()` - sposób wstrzymania zamknięcia konsoli (do momentu naciśnięcia dowolnego klawisza)

`|> ignore` - nakazuje zignorowanie zwracanego wyniku

ZADANIE 1

Wpisz swoje imię do konsoli, a następnie spróbuj je wyświetlić.

Pojawił się problem? Idę z pomocą:

Zacznijmy od tego jak definiuje się różne obiekty. Mamy do tego jedno słówko let (*de facto* jest ich więcej, ale let jest najważniejszy)

```
let a = 1                          // stała typu int
let s = "Hello World"              // stała typu string
let mutable zmienna = 2;           // zmienna typu int
let funkcja i = i+1                // funkcja typu int->int
```

Jak widać zmienne potrzebują dodatkowego słowa mutable (modyfikowalny) w deklaracji. Funkcje natomiast deklaruje się tak jak stałe, ale mają parametry. Typ funkcji zawiera ->, które pokazują kolejne argumenty przekazywane do funkcji, a na końcu zwracaną wartość. Takie coś pozwala na częściowe aplikowanie funkcji, np:

```
let skomplikowanaFunkcja a b c d = (a + 2 * b) ** (c/d)
```

```
let sF57 = skomplikowanaFunkcja 5 7
```

Najpierw zadeklarowaliśmy skomplikowanaFunkcja z czterema parametrami, a następnie przypisujemy do sF57 tę funkcję tylko z dwoma parametrami. Co się dzieje? sF57 staje się funkcją przyjmującą dwa parametry (c i d).

ODP. Zad1

ZADANIE 2

Dopisz do zadania pierwszego swoje nazwisko, a następnie wyświetl imię i nazwisko w jednym wierszu.

ODP. Zad2

„Wcięcia” (przesunięcia kolejnego wiersza względem poprzedniego) w języku F# pełnią rolę „zagnieżdżenia”.

F# grupuje instrukcje w bloki stosując właśnie wcięcia tekstu, w innych językach używane są słowa kluczowe begin – end lub nawiasy, np. {}

FUNKCJE

Funkcje definiuje się za pomocą słowa kluczowego let. Definicja funkcji ma następującą składnię -

```
let [inline] function-name parameter-list [ : return-type ]
= function-body
```

gdzie:

- *function-name* jest identyfikatorem reprezentującym funkcję.
- *parameter-list* wyświetla listę parametrów oddzielonych spacjami. Można również określić jawny typ dla każdego parametru, a jeśli nie jest określony, kompilator ma tendencję do dedukowania go z treści funkcji (jak zmienne).
- *funkcja-body* składa się z wyrażenia lub złożonego wyrażenia składającego się z wielu wyrażeń. Ostateczne wyrażenie w treści funkcji jest wartością zwracaną.
- *return-type* jest dwukropkiem, po którym następuje typ i jest opcjonalny. Jeśli typ powrotu nie jest określony, kompilator określa go na podstawie końcowego wyrażenia w treści funkcji.

ZADANIE 3

Napisz program do obliczania objętości walca, gdy promień i długość są podane jako parametry.

ODP. Zad3

ZADANIE 4

Napisz funkcje obliczające pola i obwody figur:

a. Kwadratu
b. Prostokąta
c. Koła

Funkcje mają operować na liczbach typu float.

ZADANIE 5

Napisz funkcję obliczającą wartość wielomianu $(-5y^4 + \frac{1}{3}x^2 + 3y - 7)$, dla $x, y \epsilon R$

Czy wiesz, że:

Wyniki jednych funkcji możemy przekazywać do innych, wykorzystując operatory „>>” i „|>”. Funkcje w F# mogą składać się z innych funkcji. W wyniku złożenia dwóch funkcji może powstać kolejna funkcja.

Działanie operatora „>>” można symbolicznie przedstawić jako:

```
let (>>) f g x = g(f(x))
```

Przetwarzanie potokowe umożliwia składanie funkcji, które mogą być łączone jak kolejne operacje. Operator „|>” bierze argument x i przekazuje go do funkcji f. Dzięki temu operatorowi możemy przekazywać wynik jednej funkcji do drugiej, jako jej argument.

```
let (|>) x f = f x
```

Przykład

```
open System

let liczba = 10
let funkcja1 x = x + 1
let funkcja2 x = x * 2

let wynik = (funkcja1 >> funkcja2) liczba
printfn "%d" wynik

let wynik2 = liczba |> funkcja1 |> funkcja2
printfn "%d" wynik2

Console.ReadKey() |> ignore
```

ZADANIE 6

Napisz funkcje rekurencyjną obliczającą n-ty wyraz ciągu $a_n = a_{n-1} * q$, dla a_1 i q będących argumentami funkcji

STRUKTURY DANYCH

Łańcuchem nazywamy ciąg znaków (może być pusty) ujęty w cudzysłowy.

Sekwencje modyfikujące zaczynają się od znaku „ \", a dalej występuje modyfikator.

Sekwencja	Opis
\a	Dźwięk (brzęczyk)
\b	Znak cofania
\f	Nowa strona
\n	Nowy wiersz
\r	Powrót tzw. karetki
\t	Tabulator poziomy
\v	Tabulator pionowy
\\	Jeden ukośnik wsteczny
\'	Apostrof
\"	Cudzysłów

Łańcuch znaków możemy również modyfikować: dzielić, łączyć , zamieniać określone znaki, formatować.

Formatowanie łańcuchów:

Przykład

```
open System

let imie_nazwisko = "Jan Nowak"
let n = imie_nazwisko.IndexOf(" ") //numer indeksu pierwszej spacji
let imie = imie_nazwisko.[0..2]
let nazwisko = imie_nazwisko.[4..8]

printfn "%s" imie_nazwisko
printfn "%s" imie
printfn "%s" nazwisko
Console.ReadKey() |> ignore
```

ZADANIE 7

Dla wartości x=25 i y=46 sformatuj ich wyświetlanie:

a) format liczby dziesiętnej całkowitej
b) z dodatkowymi pięcioma spacjami z lewej strony
c) uzupełnione o dodatkowe sześć zer z lewej strony

d) dodatnie znaku „+” przed wartością dodatnią
e) format liczby ósemkowej całkowitej
f) format liczby szesnastkowej całkowitej

ZADANIE 8

Dla wartości z= 56.123 sformatuj wyświetlanie:

a) wydruk liczby z w postaci wykładniczej
b) z 4 miejscami po przecinku

Struktura łańcucha formatującego jest następująca:

„%[znaczniki][szerokość][.precyzja]rodzaj”

Wartości znaczników w łańcuchach formatujących:

Znacznik	Opis
0	Wypełnienie pola liczby początkowymi zerami
-	Wyrównanie do lewej strony pola
+	Rozpoczynanie liczby od znaku (+ lub -)
spacja	Spacja przed liczbą dodatnią lub pustym łańcuchem

Rodzaje formatowania:

Rodzaj	Opis
b	Wartość logiczna true lub false
s	Łańcuch znaków
d	Liczba całkowita dziesiętna
i	j. w.
o	Liczba ósemkowa (naturalna)
u	Liczba naturalna dziesiętna (tzw. bezznakowa)
x	Liczba naturalna w zapisie szesnastkowym (małe cyfry)
X	Liczba naturalna w zapisie szesnastkowym (duże cyfry)
o	Liczba naturalna w zapisie w ósemkowym
e	Liczba zmiennopozycyjna w zapisie wykładniczym (z małym e)
E	Liczba zmiennopozycyjna w zapisie wykładniczym (z dużym E)
f	Liczba zmiennopozycyjna o podanej liczbie miejsc
F	j. w.
g	Tak samo jak e, jeżeli wykładnik jest większy od -4 lub mniejszy niż dokładność, w przeciwnym wypadku używany jest format f
G	j. w.
c	Pojedynczy znak
M	Wartość dziesiętna
A	Dowolna wartość, domyślny wygląd
%	Brak argumentu do konwersji, wynik: %

Formatowanie na inny sposób:

Przykład

```
open System
let imie_nazwisko = "Jan Nowak"
Console.WriteLine("Moje imię i nzwisko: {0}", imie_nazwisko )
let nazywam_sie = String.Format("Moje imię {0}, moje nazwisko {1}.",
"Jan", "Nowak")
Console.WriteLine(nazywam_sie)
Console.ReadKey() |> ignore
```

Co warto jeszcze znać?

INSTRUKCJA WARUNKOWA:

Format użycia:

```
if warunek_1 then
     [blok_instrukcji_1]
     wyrażenie_1
[elif warunek_2 then
     [blok_instrukcji_2]
     wyrażenie _2]
.....
[elif warunek_n then
     [blok_instrukcji_n]
     wyrażenie _n]
[else
     [blok_instrukcji]
     wyrażenie]
```

Warunki logiczne:

Operator	Opis
=	równy (błędem jest jeden znak =)
<>	nierówny (różny)
<	mniejszy
<=	mniejszy lub równy
>	większy
>=	większy lub równy

Wynikiem porównania będzie prawda lub fałsz.

Warunki logiczne można łączyć operatorami logicznymi „||” (lub) oraz „&&” (i), a także stosować zaprzeczenie „not”.

INSTRUKCJA POWTARZANIA:

Pierwszy format:

```
for licznik = start [ to | downto ] koniec do
    blok_instrukcji
```

Drugi format:

```
for licznik in sekwencja do
   blok_instrukcji
```

STRUKTURALNA OBSŁUGA WYJĄTKÓW:

```
try
 wyrażenie1
with
  | wzorzec1 -> wyrażenie2
  | wzorzec2 -> wyrażenie3
  ...
```

lub:

```
try
   wyrażenie1
finally
   wyrażenie2
```

INSTRUKCJA RAISE *– ZGŁASZAMY WŁASNE WYJĄTKI:*

```
let ex = System.ArgumentOutOfRangeException(
   "parametr","komentarz")
let wynik =
   try:
        pobranie_danych

        if x < 1 then
             raise ex

        blok_instrukcji

   with
        | :? nazwa_1 [(dane_1)] -> blok_awaryjny1
        ....
```

Dla chętnych: okno **F# Interactive**

Po uruchomieniu środowiska Visual Studio otwieramy okno pracy interaktywnej z F#. Znajdziemy je menu > View > Other Windows > F# Interactive (lub Ctrl+Alt+F).

Warto zaznaczyć, że polecenia języka F# które będę tutaj wpisywane muszą zakończyć się podwójnym średnikiem (;;) i zostać potwierdzone klawiszem ENTER. Polecenia zostaną natychmiast interpretowane przez translator języka.

Przetestujmy jego działanie, wpisując:

```
> 2+3;;
```

dostajemy:

```
val it : int = 5
```

można to przetłumaczyć jako: *wartość tego całkowita (typ int) równa 5.*

ZADANIE 9

Przetestuj wprowadzenie liczb z przecinkami dziesiętnymi i kropkami.

W celu wyczyszczenia okna pracy interaktywnej wciskamy Ctrl+Alt+R lub prawym przyciskiem myszy i wybierając z menu polecenie Reset Session.

ZADANIE 10

Przetestuj działania arytmetyczne z poniższej tabeli:

Działanie	Przykład	Odpowiedź F# Interactive
Suma	`2+3;;`	`val it : int = 5`
Różnica	`3.2-5.8;;`	`val it : float = -2.6`
Iloczyn	`5.6*3.0;;`	`val it : float = 16.8`
Iloraz	`6.8/2.3;;`	`val it : float = 2.956521739`
Potęga (x^y)	`4.5**2.4;;`	`val it : float = 36.95813377`
Modulo	`11%3;;`	`val it : int = 2`

I inne operatory: https://www.tutorialspoint.com/fsharp/fsharp_operators.htm

Odpowiedzi:

ODP. Zad1

```
open System
printf "Podaj swoje imię: "
let a = Console.ReadLine()
printfn "Twoje imię: %s" a                /%s bo ma wyświetlić string
Console.ReadKey() |> ignore
```

ODP. Zad2

```
open System
printf "Podaj swoje imię: "
let a = Console.ReadLine()
printf "Podaj swoje nazwisko: "
let b = Console.ReadLine()
printfn "Twoje imię i nazwisko: %s %s" a b
Console.ReadKey() |> ignore
```

ODP. Zad3

```
open System
let cylinderVolume radius length : float =
   // function body
   let pi = 3.14159
   length * pi * radius * radius
let vol = cylinderVolume 3.0 5.0
printfn " Volume: %g " vol
Console.ReadKey() |> ignore
```