Spartkiade 2017

F# Workshop

Übungen

# Vorausseztungen

## Windows

* Visual Studio 2015 mit F# installiert mit F# Powertools
* Visual Studio Code + F# Ionide Package

# Erste Schritte

* Lade die Resourcen für die Übungen auf deinen Rechner
* Öffene das Solution FsharpSpartakiade2017.sln
* Führe die Kompilierung durch
* Akzeptiere

# Modul 1

* Kommentare
* Ausdrucke
* Funktionen
* Rückgabe
* Einrückung

In Visual Studio, gehe zur Anwendung Modul1 und öffne die Datei ErsteSchritte.fs

## Übung: Kommentare

Kommentiere eine Zeile mit //

Kommentiere einen Block: Beginne den Kommentare mit (\* und beende diesen mit \*)

### Übung: Ausdrücke

Definiere den Ausdruck a mit Wert 1

let a = 1

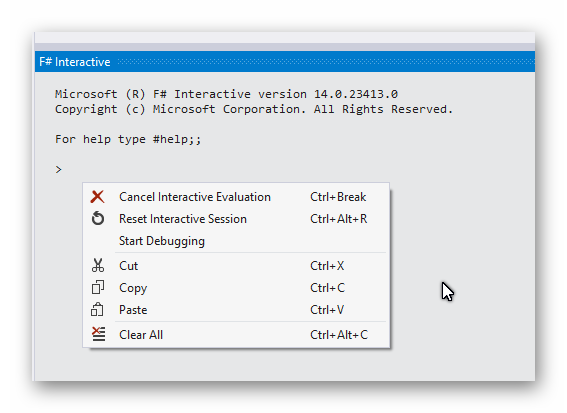
Bewege die Maus auf den Ausdruck a, der Typ wurde vom Compiler ermittelt.

Erzeuge weitere Ausdrücke mit folgenden Typen

* Ausdruck b: float mit dem Wert 1,0
* Ausdruck c: String mit dem Wert „Zeichenkette“
* Ausdruck d: bool mit dem Wert true

## Übung: Interactive

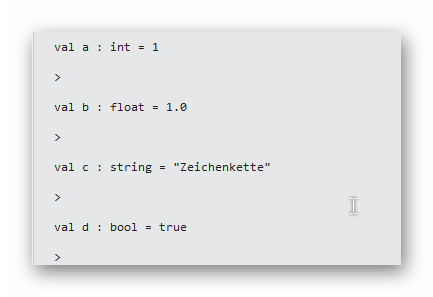
* Öffne das Interaktive Fenster
* Den Inhalt der Sitzung kann man mit rechter Maustaste zurücksetzen und neu beginnen



Ich kann jetzt die einzelnen deklarierten Ausdrücke im öffnen REPL laden.

* Zeile markieren
* Alt+Enter

Bitte dies für die die Ausdrücke a,b,c,d machen und den Inhalt des Interactive kontrollieren



Ich kann den Ausdruck in der Datei ändern und den neuen Wert erneut zum Interactive „Schicken“. Bitte probieren.

## Übung: Funktion

### Erste Funktion

In Visual Studio, gehe zur Anwendung Modul1 und öffne die Datei Funktion.fs

Erstelle eine Funktion mit folgenden Eigenschaften

* Name: add1
* Eingabe: ein Parameter x
* Berechnung: Addiere 1 zum Parameter 1

let add1 x = x + 1

Bitte beachten: Ich brauche kein Return. Der letzte Wert wird zurückgegeben.

Funktion zum Interactive schicken und Ausgabe kontrollieren

val add1 : x:int -> int

Wir erweitern die Funktion um einen Zwischen Parameter und führen eine „explizite“ Rückgabe aus.

let add1 x =

let z = x + 1

z

Bitte beachten: keine Klammern sind notwendig um die Funktion einzugrenzen.

Bitte beachten: lediglich die Angabe des Wertes z definiert diesen als Rückgabe Wert.

### Noch eine Funktion

In Visual Studio, gehe zur Anwendung Modul1 und öffne die Datei Funktion.fs

Erstelle eine Funktion mit folgenden Eigenschaften

* Name: isEven
* Eingabe ein Paramter x
* Berechnung: ist x gerade?

Bitte beachten: = ist sowohl Zuweisungsoperator als auch Gleichheitsoperator.

let isEven x = x % 2 = 0

In Visual Studio, gehe zur Anwendung Modul1 und öffne die Datei Funktion.fs

Erstelle eine Funktion mit folgenden Eigenschaften

* Name: isEven
* Eingabe ein Paramter x
* Berechnung: ist x gerade?

Bitte beachten: = ist sowohl Zuweisungsoperator als auch Gleichheitsoperator.

let isEven x = x % 2 = 0

## Übung: Script und Interactive

In Visual Studio, gehe zur Anwendung Modul1 und öffne die Datei try.fsx

* Führe einen Reset in Interactive
* Schicke den Inhalt der Datei zum Interactive
* Zeige die Ausdrücke aus ErsteSchritte.fs im Interactive. Hierfür den Namen des Ausdrucks eingeben gefolgt von zwei Semikolons.
* Definiere einen Ausdruck aPlus1. Ausdruck führt die Funktion add1 mit a (aus ErsteSchritte.fs) als Parameter.

let aPlus1 = add1 a

Schicke aPlus1 zum Interactive. Kontrolliere das Ergebnis:

val aPlus1 : int = 2

Bitte Beachten: Ich kann das Interactive Fenster immer neuladen

## Übung: Listen Funktionen

Diese sind in .net auch vorhanden! Um die Sigantur einer Funktion erhalten gibt es zwei Wege

* Funktion in einer Code Datei oder Script schreiben und F12 Taste klicken. So gelange ich in eine Deklarationsdatei wo die Signatur und das Kommentar sind
* Funktion in Interactive schreiben, mit ;; ausführen

So kann ich z.B. sehen dass die Funktion filter in Modul List folgende Signatur hat:

val filter : predicate:('T -> bool) -> list:'T list -> 'T list

val map : mapping:('T -> 'U) -> list:'T list -> 'U list

Bitte beachten: List hat viel mehr Funktion wir kommen noch dazu.

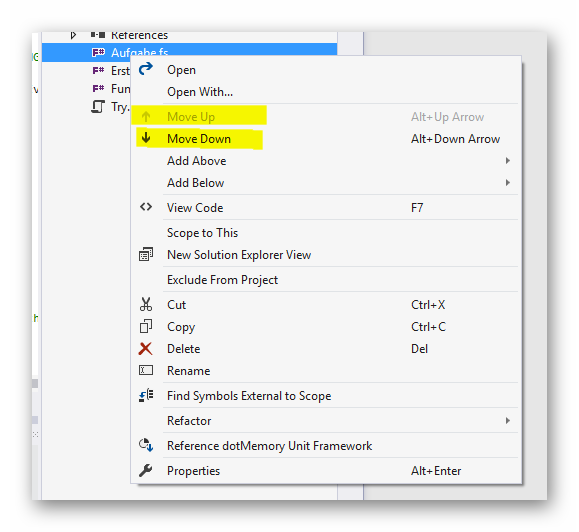
## Übung: Verketten

In Visual Studio, gehe zur Anwendung Modul1 und öffne die Datei Aufgabe.fs. Darin ist die Beschreibung der Aufgabe.

Erstelle einen Ausdruck der den gewünschten Wert beinhaltet:

* Analysiere die notwendigen Referenzen
* Positioniere die Datei Aufgabe so dass diese an die notwendigen Referenzen herankommen kann
* Referenziere die notwendigen Code Dateien

Die Lösung

* Die Funktionen die ich brauche sind add1 und isEven
* Beide befinden sich in der Datei Function.fs
* Von daher muss Aufgabe.fs „unter“ Function.fs stehen
* In VS, Aufgabe.fs wählen und die rechte Maustaste klicken.
* 
* Bewege die Datei Aufgabe.fs bis diese die notwendigen Funktionen erreichen kann
* Importiere die Funktionen mit Hilfe des Schlüssel Wortes open
* Definiere einen Ausdruck summeOhneVerkettung der die Werte ermittelt ohne Verwendung des Verkettungsoperator
* Definiere einen Ausdruck summe der die Werte ermittelt mit Hilfe des Verkettungsoperator
* Teste den Wert aus dem Script Try.fsx heraus.