2.Einführung Ocaml Prog

Hint:

Bei Instalation darf man nicht mit Admin-Privillige ausführen, stattdessen nur mit derzeitig verwendetem Benutzer.

Normalweise klonen Arbeit von Repo und dann Ordner richten

Dann im WSL eingeben dune build, um Felher temporär zu lösen

Manchmal wird Fehler „Failwith „TODO““ ausgedruckt, dass ich normal und richtig, da du Nichts geschafft hast.

Wenn du fertig implementiert und debuggen willst, sollst du zuerst Änderung speichern und dann noch einmal dune build eingeben, um Code fertig zu kompilieren. Danach kannst du nach dune utop src debuggen

**2 Basics**

**• Interpreter Environment**

**• Expressions**

**• Definitions of Values**

**• More Complex Datatypes**

**• Lists**

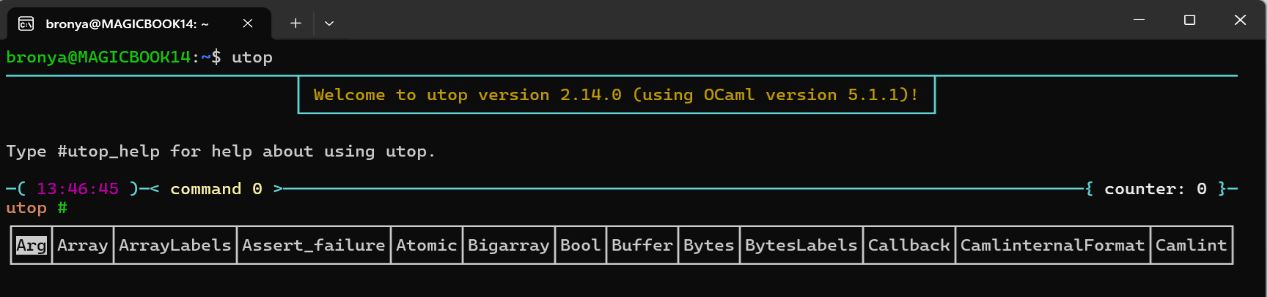
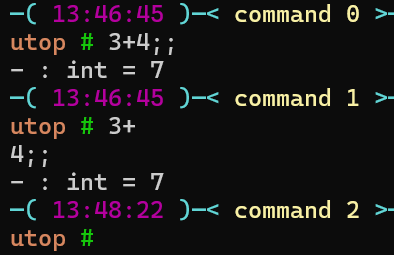
**• Definitions (cont.)**

**• User-defined Datatypes**

**2.1 BasisUmfeld**

BasisInterpreter ist Ocaml, in dem gibt es insg 2 Datei, ein ist mit „ml“, wo dein Code speichert, und andere ist „mli“ wo Funktion Deklarierung besteht.

Bei Ocaml Enviroment (Beispielsweise utop) :



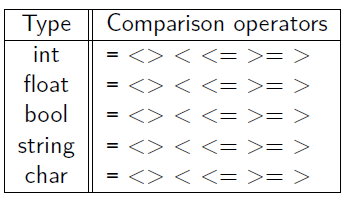
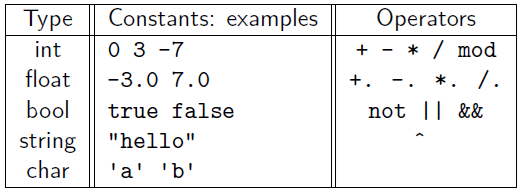
kann man Ausdrucksform eingeben und dann Ergebnis rückgeben.

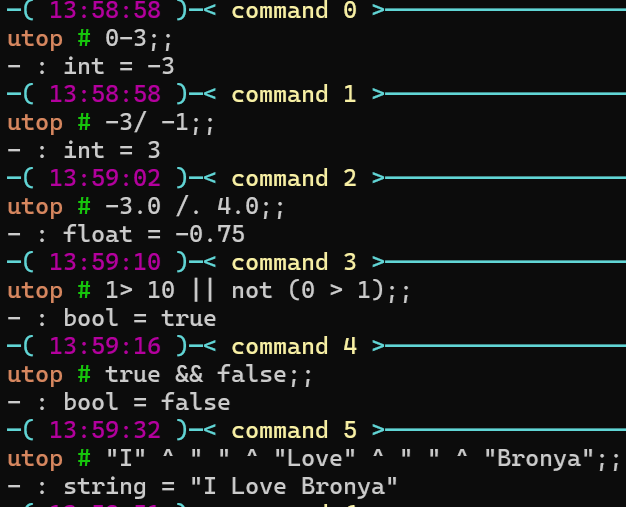
**2.2 Ausdrucks**

Anfangs Eingabe Feld mit **#**und man kann **in einer Zelle oder mehreren Zellen** tippen, **Nur** man **;;** tippt wird Eingabe beenden.

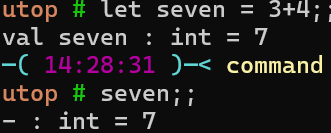
Ergebnis wird **zusammen kalkuliert** und **mit Datentyp** rückgeben. (**Implizit Datentyp Berechnung**)

Hier sind einige Vordefinierte Operatoren:



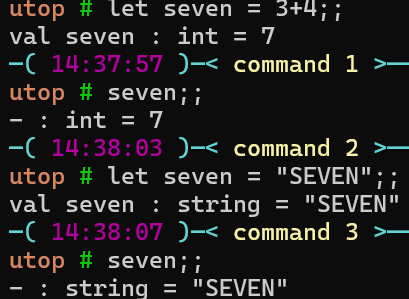
Bsp.:

So wie Foto gezeigt, Datentyp wird nicht abhängig eingeben aber, wenn **unerwartete Variable** eingegeben, wird Fehler auswerfen (**Cast Err Int**)!!!

**2.3 Variable Definition**

Durch Schlüsselwort **let** kann man variable definieren.

Variable behält gegebene Wert immer!!!

(Wie Beispielsfoto gezeigt, zuerst man Variable mit Name „seven“ definiert und dann Als Ergebnis von „3+4“ erhalten, So wird im Konsol „val seven . int = 7 “ gedruckt.

Wenn man noch einmal seven aufruft, wird gleiche Wert rückgegeben.)

Das kann man so verstehen, falls man erneut mal ein Variable mit gleichem Namen zuwiesen wird, erhält eine neue Variable, trotzdem mit gleichem Namen und mögl. mit gleichem Datentyp, die Beide sind **absolut unterschiedlich**!!!!!  
Und Alte kann **nicht mehr aufgerufen** werden!!!!

**Bei Benennen einer Variable darf man nur mit Kleine Buchstabe anfang!!!**

**2.4 Mehre komplexe Datentypen:**

**2.4.1Paar (Pairs):**

Ein: # (3,4);;

Aus: -: **int \* int** = (3,4)

Ein: #(1 = 2， “Bronya”);;

Aus: -: **bool \* string** = (false, “Bronya“)

**2.4.2 Tupel:**

Ein: # (2, 3, 4, 5);;

Aus: -: **int \* int \* int \* int** = (2, 3, 4, 5)

Ein:# (“Seele“, true, 1145141);;

Aus: -: **string \* bool \* float** =(“Seele“, true, 1145141)

**2.4.3Gleichzeitig definierte Variable:**

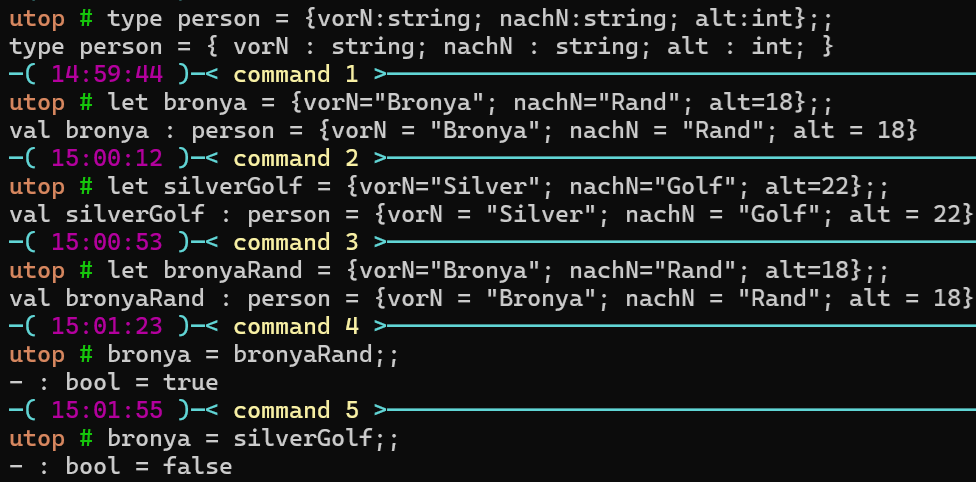
Ein: #let (x,y) = (3, 4.0);;

Aus: -: **val x: int = 3**

**val y: float = 4.0**

Ein: #let(\_, y) = (3, 4.0);; //”\_” ist eine anonyme Variable, bedeutet gleiche Form besitzt, aber Wert unwichtig

Aus: -: **val y: float = 4.0**

Beispiel:

Records sind Tupel mit benannten Komponenten, deren Reihenfolge daher irrelevant ist.

Als neuer Typ muss ein Record vor seiner Verwendung mittels einer Typdeklaration eingeführt werden.

Typnamen und Record-Komponenten beginnen mit einem **Kleinbuchstaben**.

Als ein Objekt, man kann auch seiner Eigenschaften aufrufen:

(Durch Auswahl von Komponenten)

Man kann auch Mustervergleichen (Patternmatching):

Ein: # let {vorN = x; nachN = y: alt = z} = bronya;;

Aus: -: **val x : string = Brnoya**

**val y: string = Rand**

**val z: int = 18**

Falls wir nur für einige der Alle interessiere, kann andere mit Anonymen markieren.

Ein: #let {vorN = x; \_ };;

Aus: -: **val x ; string = „Bronya“**

**2.4.4 Match and If**

match choose with

| 0 -> "null"

| 1 -> "ONE"

| \_ -> failwith "Uncountable"

match boolean with

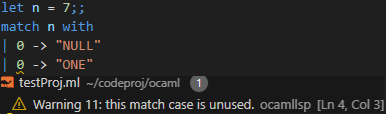
| true -> e1

| false -> e2

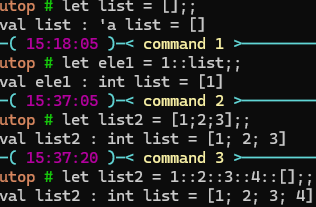
//Hier kann man so umsetzen: if boolean then e1 else e2

**Vorsicht auf redundante und unvollständige Übereinstimmungen!!!注意冗余和不完整匹配！！！**

不完整匹配

匹配冗余（一量多用）

**2.5 Lists**

**2.5.1 Def:** Lists sind durch **[] und ::** definiert

Bsp.: wie Foto rechts gezeigt

**Obligatorisch: Alle Elemente haben gleiche Datentyp**

**2.5.2 List mit beliebigem Datentyp:**

**Alpha list** beschreibt List mit Elementen von Datentyp alpha

Beispielsweise 'a list

**[]** beschreibt **leere List für beliebige Datentyp**

**2.5.3 Matching von Liste**

match list with

| [] -> -1

| x::xs -> x;;

**2.6 Def Funktion**

Ein: # let func x = 2 \* x;; // func: name von Funktion, Nachdem Funktionname folgt Parametern

Aus: -: **int -> int = <fun>** //Funktionname ist nur ein variable, dessen Wert eine Funktion ist

Alternativ können wir eine Variable einführen, deren Wert eine Funktion ist.

Ein: let varName = funkName x -> 2 \* x;;