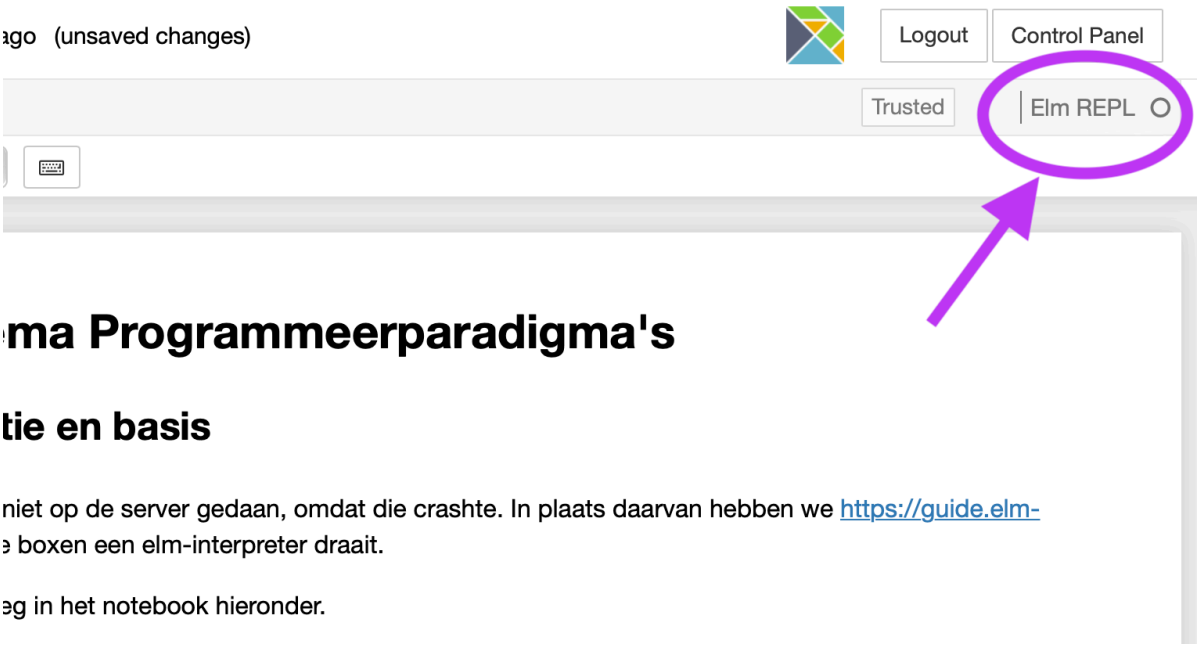
Docentenscholing keuzethema Programmeerparadigma's

Workshop 10 juni 2020 – Introductie en basis

Het hands-on gedeelte van deze workshop hebben we niet op de server gedaan, omdat die crashte. In plaats daarvan hebben we https://guide.elm-lang.org/core_language.html (https://guide.elm-lang.org/core_language.html) gebruikt, waar in de zwarte boxen een elm-interpreter draait.

De oefeningen die we daar gedaan hebben staan ruwweg in het notebook hieronder. Commentaar staat meestal *na* het voorbeeld.

Dit is een Elm REPL notebook. Zie pijl.



Values en expressions

In [1]:
```
2 + 2
```

```
4 : number
```

Type regels elm in een cell, executeer met **Ctrl-Enter** of **Shift-Enter**.

In [2]:
```
"ape" ++ "brood"++"boom"
```

```
"apebroodboom" : String
```

Om operatoren hoeven geen spaties te staan, maar het is goede stijl om dat wel te doen.

Definties

In [3]:
```
x = 3
```

```
3 : number
```

In [4]:
```
naam = "Coen"
```

```
"Coen" : String
```

In dit notebook wordt na de evaluatie van een cel het resultaat en het type van het resultaat gezet.

Er is geen onderscheid tussen variabelen (namen van waarden) en functies. Je definieert ze met =.

In [5]:
```
greet2 name = \
    "Dag, " ++ name ++ " " ++ name
```

```
<function> : String -> String
```

Let op de \ achter iedere regel (behalve de laatste) in de cel. Dit is alleen nodig in deze Elm-REPL notebooks.

In [6]:
```
greet2 "Coen"
```

```
"Dag, Coen Coen" : String
```

In [7]:
```
cijfers = [7, 5, 8, 8, 4]
```

```
[7,5,8,8,4] : List number
```

In [8]:
```
List.length cijfers
```

```
5 : Int
```

Functies zijn ingedeeld in packages. De naam van het package, zoals List, moeten vóór de functienaam staan. Je kunt packages importeren zodat dat niet nodig is, maar dat werkt niet in deze Elm-REPL notebooks.

Recursie

In [13]:
```
krikopFout cs = \
    case cs of \
    [] -> [] \
    c :: ds -> (c+1) :: krikop cs
```

```
<function> : List number -> List number
```

In [14]:
```
krikopFout cijfers
```

```
[8,8,6,9,9,5]
    : List number
```

In [15]:
```
krikop cs = \
    case cs of \
    [] -> [] \
    c :: ds -> (c+1) :: krikop ds
```

```
<function> : List number -> List number
```

In [16]:
```
krikop cijfers
```

```
[8,6,9,9,5] : List number
```

Definieer een functie die het verhogen van één cijfer wat subtieler doet.

In [17]:
```
increment n = \
    if n < 10 then n + 1 \
    else n
```

```
<function> : number -> number
```

In [19]:
```
increment 5
increment 10
```

```
6 : number
10 : number
```

De functie **map** uit List voert een functie uit op alle elementen van een lijst. Functie en lijst zijn beide parameters van de functie.

In [20]:
```
List.map increment cijfers
```

```
[8,6,9,9,5] : List number
```

In [21]:
```
List.map increment (List.map increment cijfers)
```

```
[9,7,10,10,6] : List number
```

In [22]:
```
List.map increment cijfers |> List.map increment
```

```
[9,7,10,10,6] : List number
```

|> stuurt het resultaat van het linkerdeel als input naar het rechterdeel. Zoals een pipe, **|**, in de Unix shell. Scheelt haakjes en laat je de fucnties in de logsiche volgorde opschrijven.

[Niet in de workshop]. Je kunt de herhaling van de operatie ook in het functie-argument van List.map laten plaatsvinden. **>>** is functie-compositie. (rondje in de wiskunde, maar die werkt in de omgekeerde volgorde; wil je precies die, dan gebruik je **<<**)

In [23]:
```
List.map (increment >> increment) cijfers
```

```
[9,7,10,10,6] : List number
```

Twee concatenatie-operatoren

++ schakelt twee lists of strings aan elkaar

:: zet een *element* voor een lijst

In [27]:
```
[1] ++ [2,3]
1 :: [2,3]
```

```
[1,2,3] : List number
[1,2,3] : List number
```

In [28]:
```
[1,2] ++ [3,4]
1 :: 2 :: [3,4]
```

```
[1,2,3,4] : List number
[1,2,3,4] : List number
```

In [ ]: