Discussion 1

August 30

Agenda

- Some first day remarks
  - Remarks about the course
  - Introductions
- Review of expressions vs. values
- Hazel walkthrough

About EECS 490

- Some negative remarks
  - “Programming Languages” i.e. “Theory of Programming Languages”
  - We will ***not*** survey programming languages/paradigms
    - EECS 390: Programming Paradigms
  - We will ***not*** talk about OOP (although TAPL has some nice material on it)
    - TAPL = Benjamin Pierce’s *Types and Programming Languages*, a supplementary reading

About EECS 490

- Some positive remarks
  - We ***will*** talk about functional programming and imperative programming
  - We ***will*** use Hazel, OCaml, and Rust

Logistics

- Discussion Section
  - Friday 12pm-1pm in 1200 EECS

Logistics

- Assignments
  - Weekly assignments (generally) will be due Fridays @ 6:00 pm ET
  - Usage of a late day extends the submission deadline by one *business* day, i.e. (usually) until the following Monday @ 6:00 pm ET
  - Only **one** late day per assignment
  - Assignment solutions are released right after the late deadline
    - This is why only one late day is allowed per assignment
  - You have **three late days** for the semester

Logistics

- Assignments
  - Assignment solutions are released after the late deadline, so we ***cannot offer assignment extensions***
  - **Please** reach out to the course staff for exceptional reasons (physical and mental health, etc.)
- This is a summary of the course syllabus! Please actually read the syllabus for the finer details
- A1:
  - Releasing next Tuesday (September 3)
  - Due the following Friday (September 13)

Introductions

- Hi! I’m Gregory :)
- I’m from Knoxville, Tennessee
- PhD student studying creative support and AI interfaces
- I like playing violin and biking
- Office hours in 4440 EECS (or online)
  - Tuesday / Thursday 6pm-7pm (right after class!)
- Who are you??

What is this “Discussion” anyways?

- My goal is **NOT** for this to be a mini-lecture
- My goal **IS** to make you more comfortable with the material and vocabulary
- If you do not talk, it **WILL** be quiet and it **WILL** be a little bit awkward…

Expressions vs values

- Values **are** expressions
- Expressions **are not** necessarily values
  - Evaluating to a value doesn’t mean it’s a value

Is this an expression or value?

```
3
```

Value!

Is this an expression or value?

2+3 Expression!

Is this an expression or value?

```
1.618
```

Value!

Is this an expression or value?

```
(2, true)
```

Value!

Is this an expression or value?

```
fun x -> x+1
```

Value!

Is this an expression or value?

```
if true then
  2
else
  3
```

Expression!

Is this an expression or value?

```
if true then
  2
else
  false
```

Expression! (inconsistent branches)

Is this an expression or value?

```
let x =
  fun x -> x+1
in
x(2)
```

Expression!

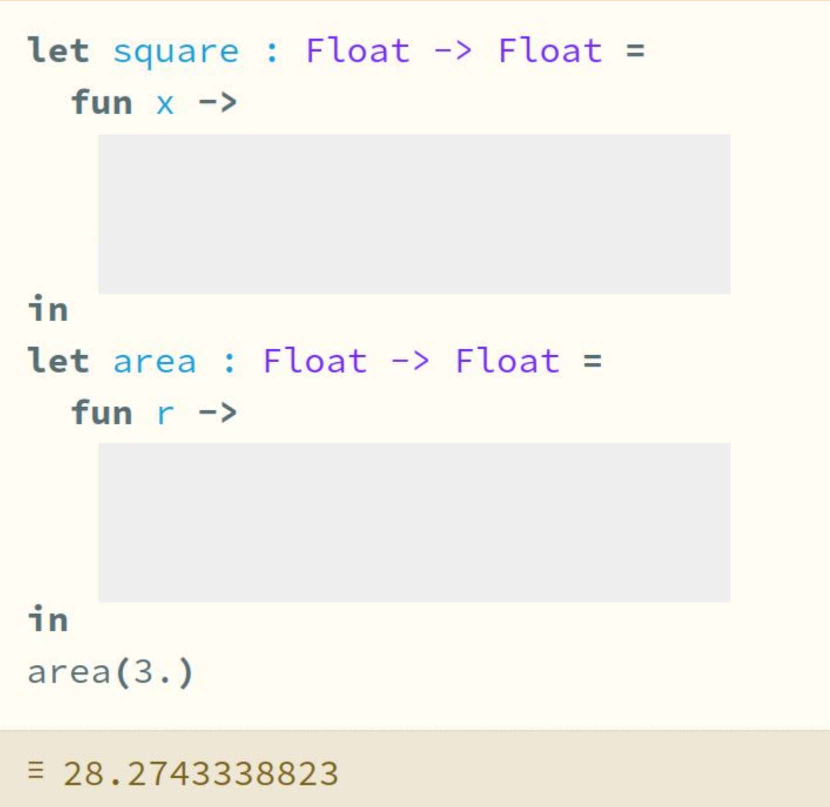
Live Demo!

- Covering:
    - Expressions and values
    - Base types and associated forms (if/then/else, numeric)
    - Let
    - Functions
    - Tuples (intro, elim)
    - Lists (intro, elim)

Live Demo!

https://hazel.org/build/dev

Exercise 1: square, area



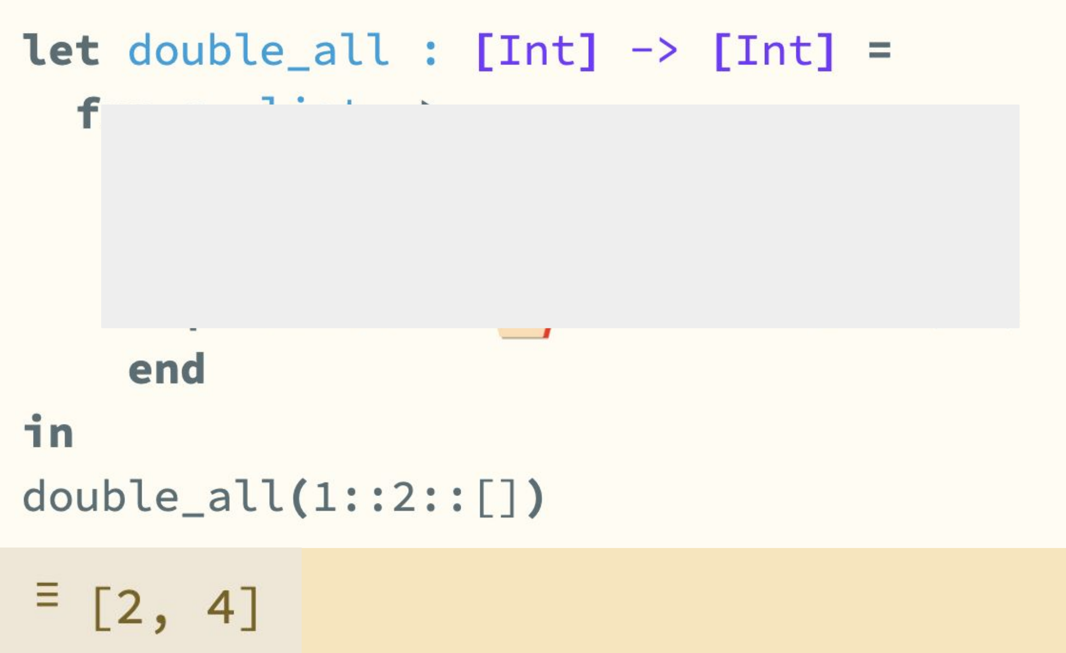
```
let square : Float -> Float =
  fun x ->



in
let area : Float -> Float =
  fun r ->



in
area(3.)
```

```
≡ 28.2743338823
```

Define functions that

1. return the square of a floating point, and
2. return the area of a circle given its radius

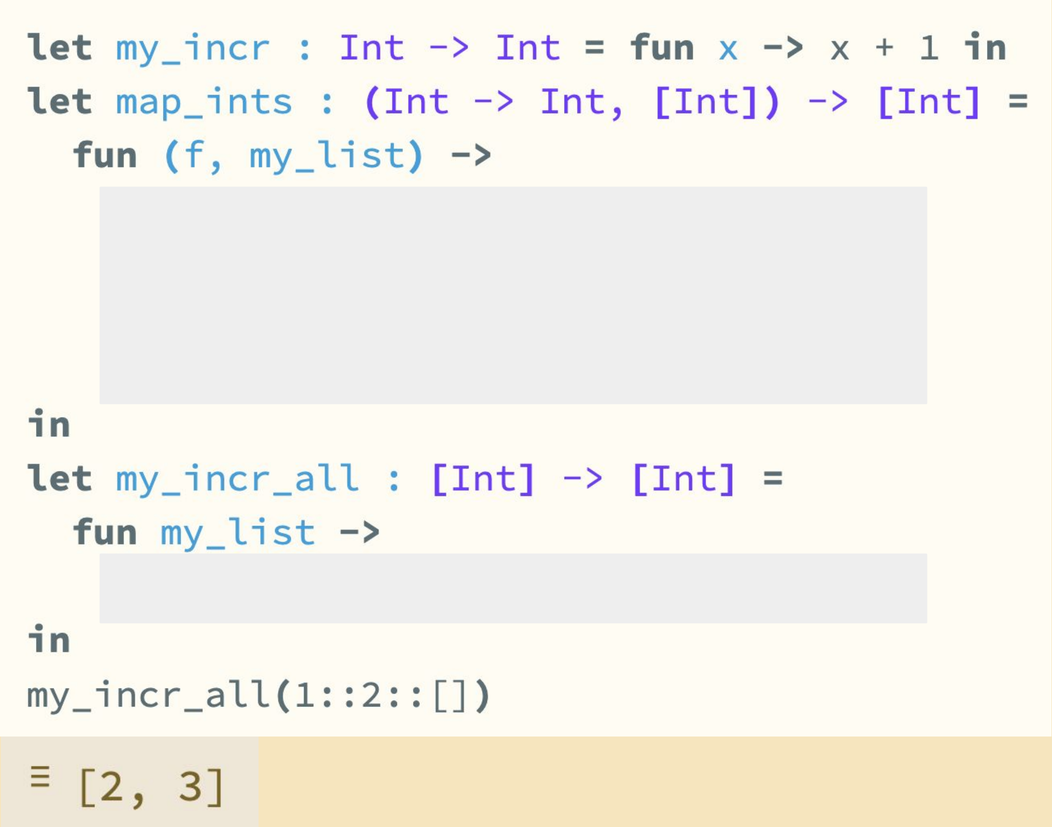
Exercise 2: double_all



```
let double_all : [Int] -> [Int] =
  f[redacted]
    end
in
double_all(1::2::[])
```

```
≡ [2, 4]
```

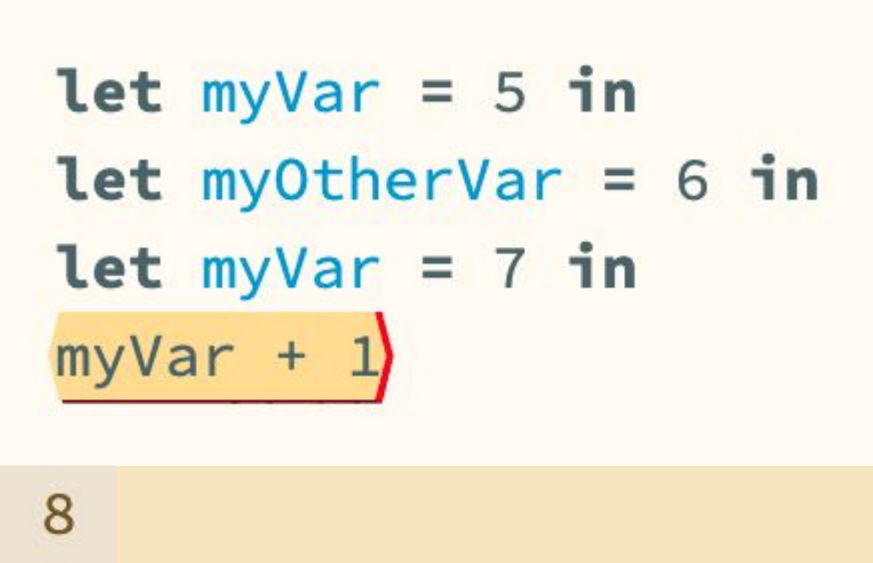
Exercise 3: my_incr_all using map_ints



```
let my_incr : Int -> Int = fun x -> x + 1 in
let map_ints : (Int -> Int, [Int]) -> [Int] =
  fun (f, my_list) ->




in
let my_incr_all : [Int] -> [Int] =
  fun my_list ->

in
my_incr_all(1::2::[])
```

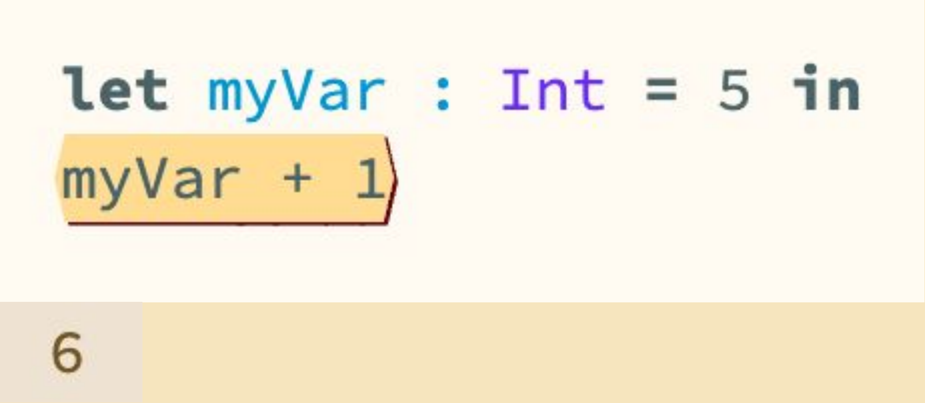
```
≡ [2, 3]
```

Let



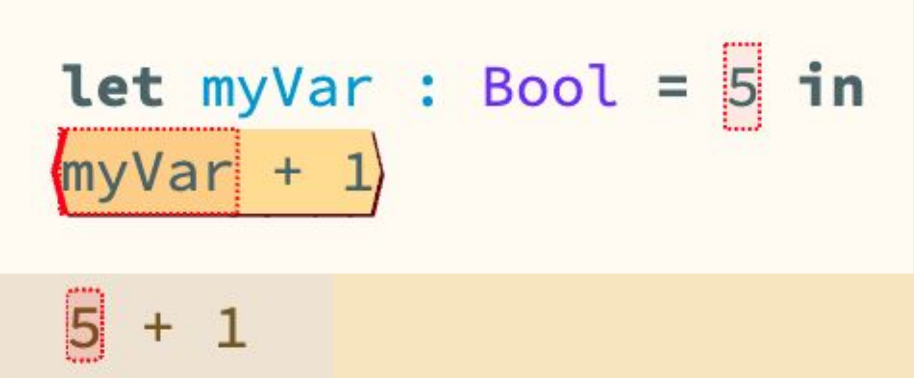
```
let myVar = 5 in
let myOtherVar = 6 in
let myVar = 7 in
myVar + 1
```

```
8
```



```
let myVar : Int = 5 in
myVar + 1
```

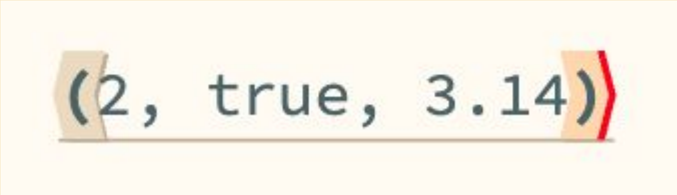
```
6
```



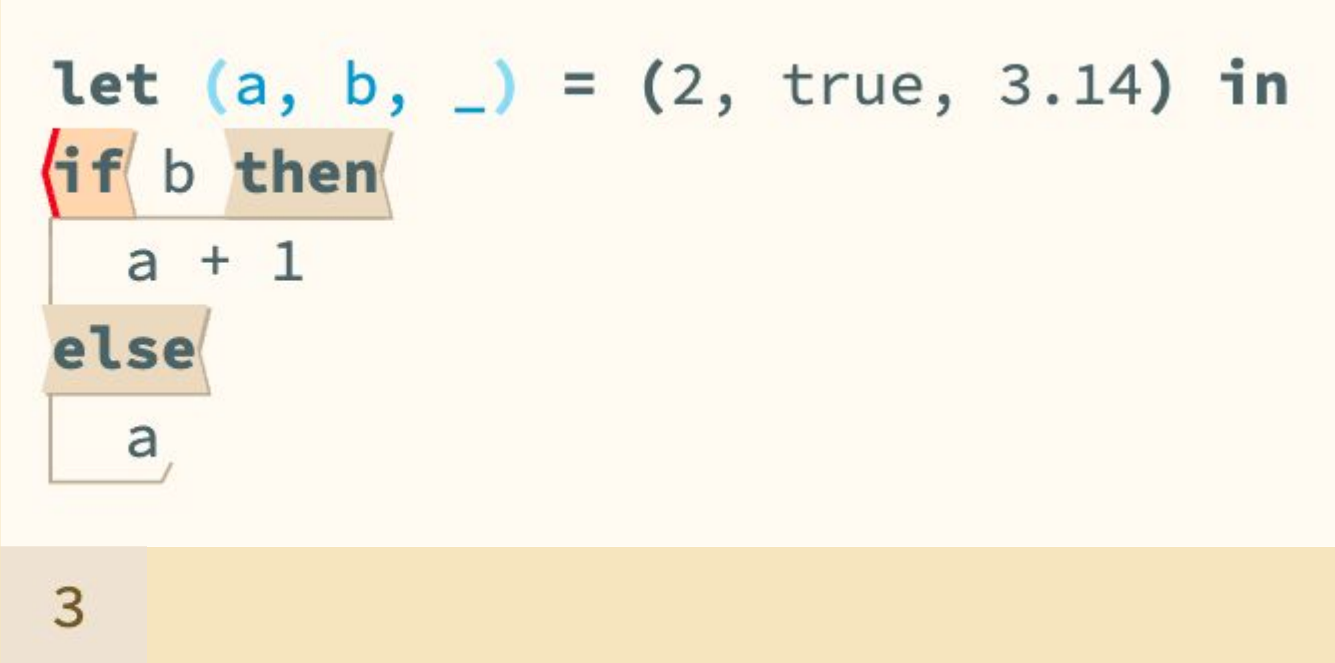
```
let myVar : Bool = 5 in
myVar + 1
```

```
5 + 1
```

Tuples



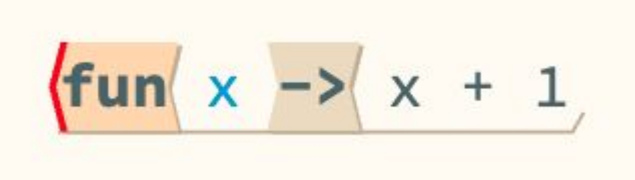
```
(2, true, 3.14)
```



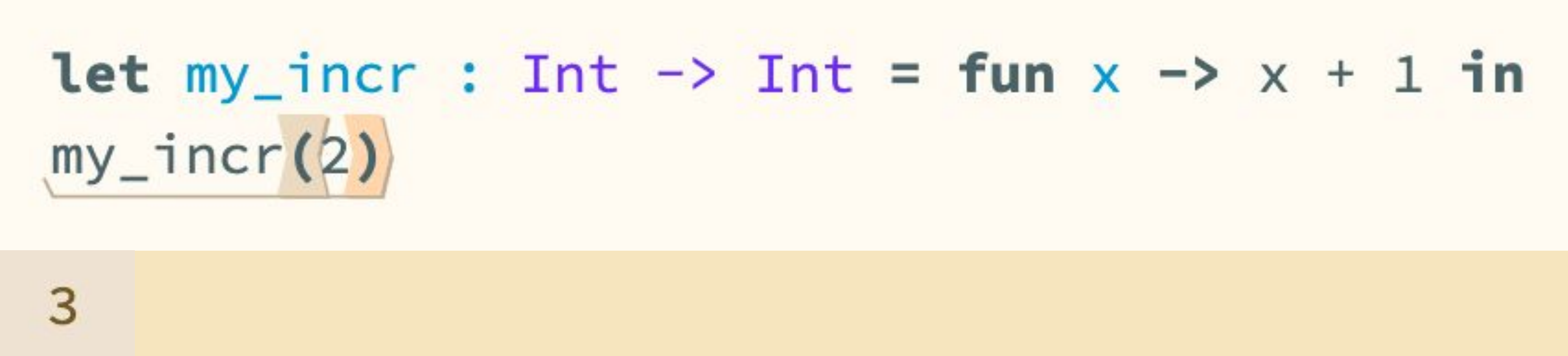
```
let (a, b, _) = (2, true, 3.14) in
if b then
   a + 1
else
   a
```

```
3
```

Functions



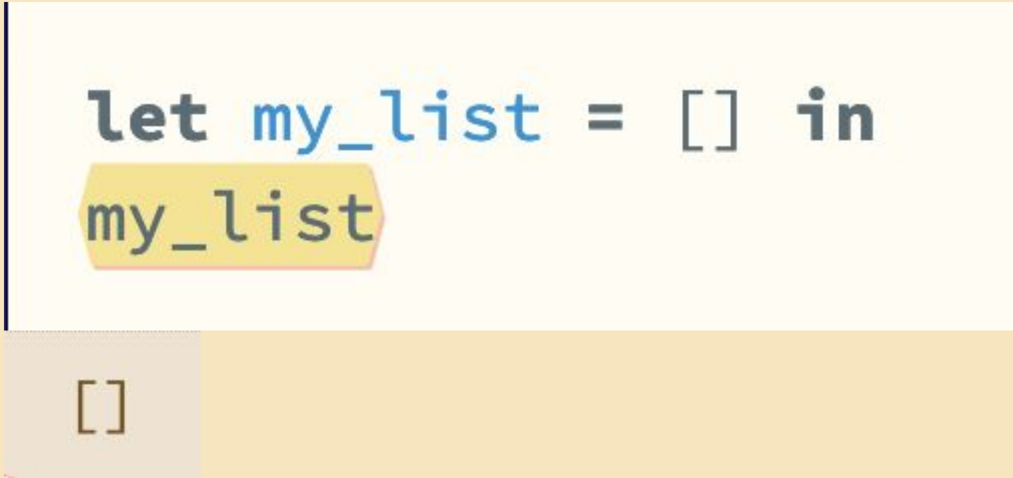
```
fun x -> x + 1
```



```
let my_incr : Int -> Int = fun x -> x + 1 in
my_incr(2)
```

```
3
```

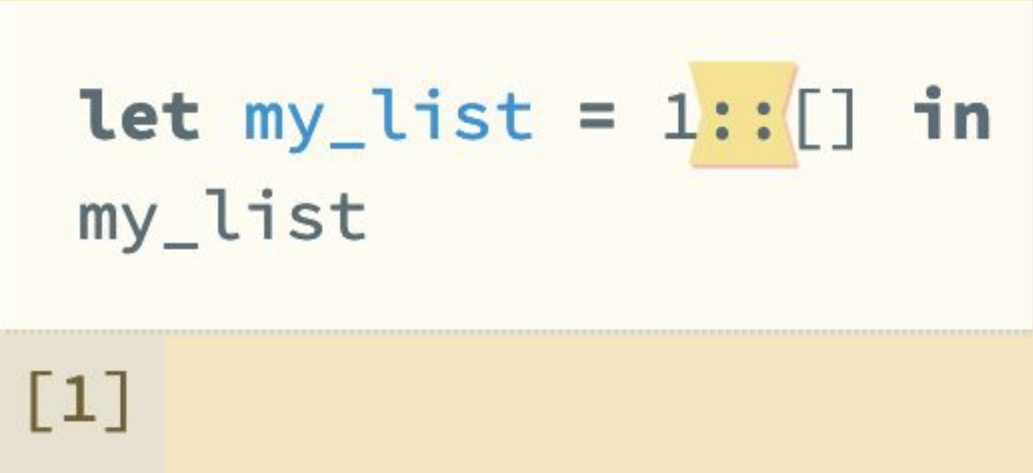
Lists



```
let my_list = [] in
my_list
```

```
[]
```

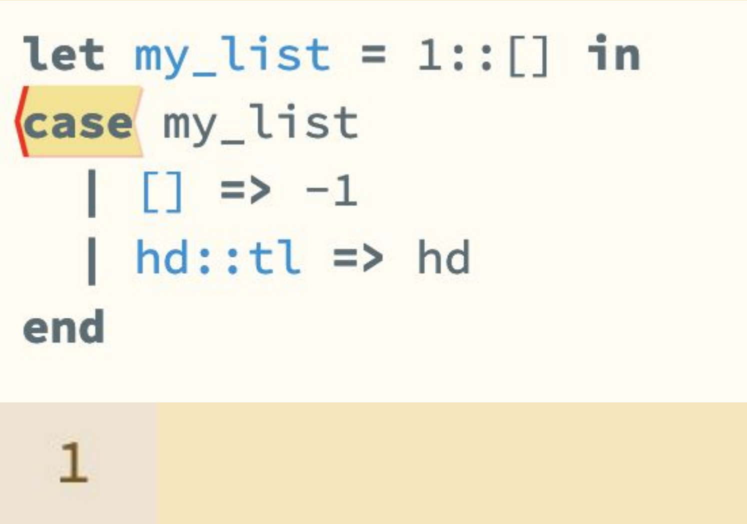
Lists



```
let my_list = 1::[] in
my_list
```

```
[1]
```

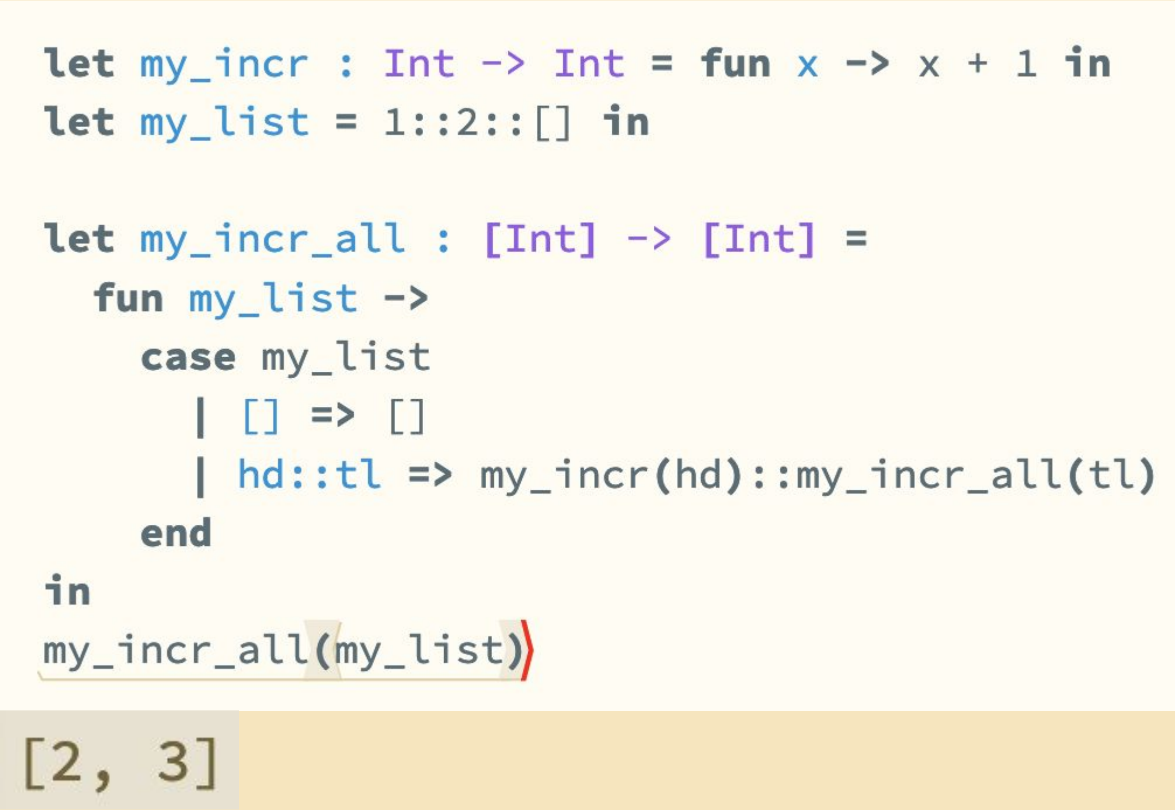
Lists



```
let my_list = 1::[] in
case my_list
  | [] => -1
  | hd::tl => hd
end
```

```
1
```

Lists



```
let my_incr : Int -> Int = fun x -> x + 1 in
let my_list = 1::2::[] in

let my_incr_all : [Int] -> [Int] =
  fun my_list ->
    case my_list
      | [] => []
      | hd::tl => my_incr(hd)::my_incr_all(tl)
    end
in
my_incr_all(my_list)
```

```
[2, 3]
```