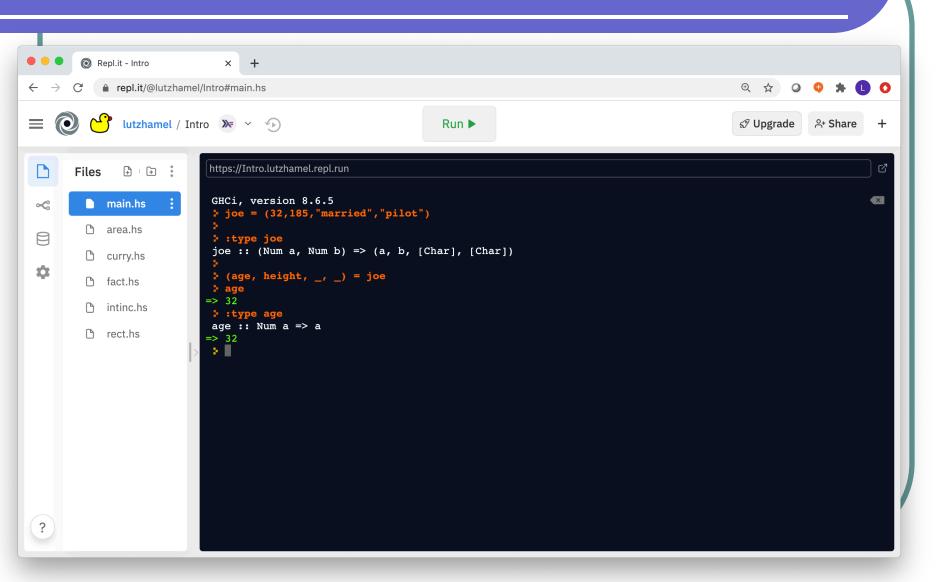
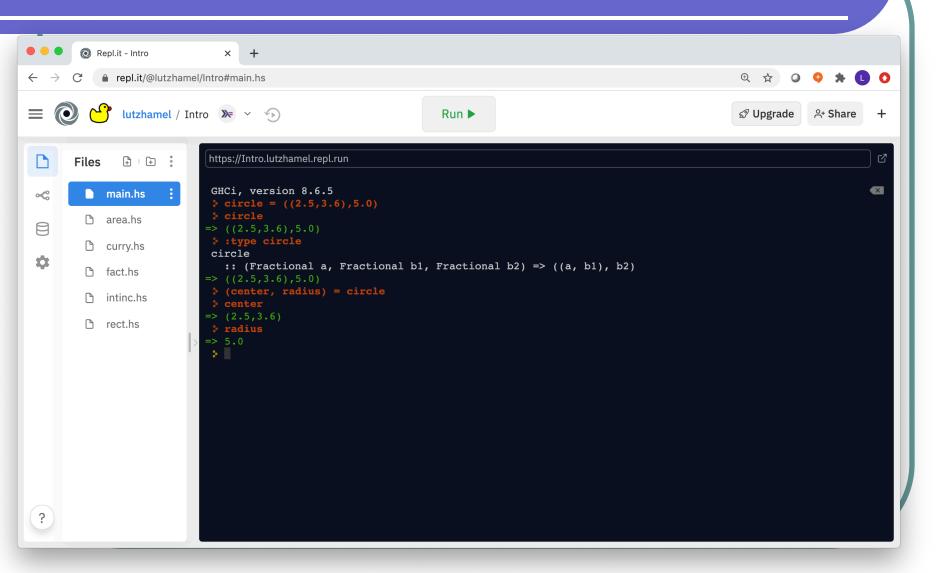
Tuples have almost the same syntax as Rust.

```
> joe = (32,185,"married","pilot")
> :type joe
joe :: (Num a, Num b) => (a, b, [Char], [Char])
> (age, height, _, _) = joe
> age
=> 32
> :type age
age :: Num a => a
Pattern Matching
```



 Nested tuples – say we want to specify a circle with a center and a radius...

```
> circle = ((2.5,3.6),5.0)
> circle
=> ((2.5,3.6),5.0)
> :type circle
circle :: (Fractional a, Fractional b1, Fractional b2) => ((a, b1), b2)
=> ((2.5,3.6),5.0)
> center, radius) = circle
> center
=> (2.5,3.6)
> radius=> 5.0
```



In a list all elements are of the <u>same type</u>

```
coddlist = [ 1, 3, 5, 7, 9 ]
coddlist :: Num a => [a]
cdifferent from tuples!
```

```
nested = [(1,2),(3,4)]

nested = [[1,2],[3,4]]

nested = [[1,2],[3,4,5]] what is the type of these constructions?

nested = [(1,2),(3,4,5)]
```

 There exists a special list → the empty list: []

```
> mylist = []
> :type mylist
mylist :: [a]
>
```

Polymorphic type!

- null tests whether a list is empty
  - null :: [a] -> Bool

```
: null [1,2,3]
=> False
: null []
=> True
: |
```

- ++ concatenates two lists
  - (++) :: [a] -> [a] -> [a]

Recall, any infix operator can be viewed as a function by putting parentheses around it.

```
[1,2,3]++[4,5,6]
=> [1,2,3,4,5,6]
: ["not"]++["married"]
=> ["not","married"]
['n','o','t']++"married"
                                     But...
=> "notmarried"
```

- : (cons operator) glue elements together to form a list
- the last elements has to always be a list
  - (:) :: a -> [a] -> [a]

```
=> [1,2,3]
    : 2 : 3 : [] == [1,2,3]
  1 : rest = [1,2,3]
    : rest = [4,5,6]
   Exception: <interactive>:26:1-18: Non-exhaustive patterns in 1 : rest
  a : 2 : rest = [3,2,1,0]
  [1,0]
```

- head return the <u>first</u> element of a list
  - head :: [a] -> a

- tail return the list without its first element
  - tail :: [a] -> [a]

```
> tail [1,2,3]
=> [2,3]
> tail "Hello"
=> "ello"
> x = tail []
> x

*** Exception: Prelude.tail: empty list
> |
```

(f) y = 1::2::3

```
(a) x = ["hello"] ++ ["there"]
(b) x = ["hello" ++ "there"]
(c) joe = (32, 185, "married", "pilot")
   jack = (29, 160, "not married", "cook")
   people = [joe, jack]
(d) I = [[1,2,3],["one", "two", "three"]]
(e) x = [1,2,3]
   h = head(x)
   t = tail(x)
   I = h : t
   1?
```