Programming 2
Tutorial 3

Functional Programming

Nedim Šišić

2023

Examples

- fun x -> x + 1;;
  - : int -> int = <fun>
- fun x -> x;;
  - : 'a -> 'a = <fun>
- (fun x -> x + 1) 1;;
  - : int = 2

Examples

- let f = fun x -> x;;
  val f : 'a -> 'a = <fun>
- Same as:
  let f x = x;;
  val f : 'a -> 'a = <fun>
- f 1;;
  - : int = 1
- f (-1)
  - : int = -1

Examples

OCaml functions always take *a single* parameter

```
let sub a b = a - b;;
val sub : int -> int -> int = <fun>
```

→ “sub” is a higher order function

```
sub 5 4;;
- : int = 1
```

```
let subFrom5 = sub 5;;
val subFrom5 : int -> int = <fun>
```

→ “sub 5” returned a function

```
subFrom5 4;;
- : int = 1
```

Exercises

Write a function that:

- takes x and returns the square of x
- takes a 2-tuple and returns its values in a list
- takes two numbers and returns the larger one (e.g. by using the "if" statement

Exercises

Write a function that:

- takes x and returns the square of x

```
let square x = x*x;;
```

- takes a 2-tuple and returns its values in a list

```
let tupleToList (c, d) = [c; d];;
```

- takes two numbers and returns the larger one (e.g. by using the "if" statement

```
let max a b =
  if a > b then a else b;;
```

Examples – Recursion

- let rec sumToN n =
  if (n = 0) then 0
  else n + sumToN (n - 1);;

  val sumToN : int -> int = <fun>

  sumToN 4;;
  - : int = 10

- let rec createList n =
  if n = 1 then [1]
  else createList (n-1) @ [n];;

  sumToN 5;;
  - : int list = [1; 2; 3; 4; 5]

Execution of sumToN 4:

sumToN 4 = 4 + sumToN 3 = 4 + (3 + sumToN 2) = 4 + (3 + (2 + sumToN 1))
= 4 + (3 + (2 + (1 + sumToN 0))) = 4 + (3 + (2 + (1 + 0)))
= 4 + (3 + (2 + 1)) = 4 + (3 + 3) = 4 + 7 = 10

Examples – Recursion

- Handling lists
- List.tl [1];;
  - : int list = []
- # List.tl [1; 2];;
  - : int list = [2]
- # List.hd [3];;
  - : int = 3

- A function that sums all elements of a list

```
let rec sumList list =
  if list = [] then 0
  else(List.hd list) + sumList(List.tl list);;
```

Exercises

Write a function that:

- computes n! (n factorial) for integer n
- reverses a list

Exercises

Write a function that:

- computes n! (i.e., n factorial) for integer n

```
let rec factorial n =
  if n = 1 then 1
  else n * factorial (n-1);;
```

- reverses a list

```
let rec reverseList list =
  if list = [] then []
  else reverseList (List.tl list) @ [List.hd list];;
```

Examples – Pattern Matching

- let isBetween0and5 n =
  match n with
  | 0 | 1 | 2 | 3 | 4 | 5 -> true
  | _ -> false;;
- let rec sumList l =
  match l with
  | [] -> 0
  | h::t -> h + sum t;;

- Symbol "_" matches any value.
- [1; 2; 3] is matched with h::t so that:
  h = 1, t = [2; 3]
- [1; 2] is matched with h::t so that:
  h = 1, t = [2]
- [1] is matched with h::t so that:
  h = 1, t = []
- [] is not matched with h::t (it is matched with [], for example)

Exercises

Write a function that:

- checks if a given character is a vowel (‘a’, ‘e’, ‘i’, ‘o’, or ‘u’)
- finds the length of a list
- reverses a list using pattern matching
- returns the last element for a list of integers and 0 for an emtpy list

Exercises

Write a function that:

- checks if a given character is a vowel (‘a’, ‘e’, ‘i’, ‘o’, or ‘u’)

```
let isVowel c = match c with
  | 'a' | 'e' | 'i' | 'o' | 'u' -> true
  | _ -> false;;
```

- finds the length of a list

```
let rec getListLength list = match list with
  | [] -> 0
  | _::t -> 1 + getListLength t;;
```

Exercises

Write a function that:

- reverses a list using pattern matching

```
let rec reverseList list = match list with
  | [] -> []
  | h::t -> reverseList t @ [h];;
```

- returns the last element for a list of integers and 0 for an emtpy list

```
let rec getLastInteger l =
   match l with
    | [] -> 0
    | [e] -> e
    | h::t -> getLastInteger t;;
```