Going Functional

Primož Gabrijelčič
Functional programming
Functional programming

- Computation = evaluation of (mathematical) functions
- Based on lambda calculus
- No state
- No mutable data
- No side effects
Functional vs. imperative

• Imperative programming
  – Functions can have side effects

• Functional programming
  – Output depends only on the input arguments
Functional programming

• Immutable variables
• Pattern matching
• Higher-order functions
• Recursion
Functional programming in Delphi
Functional programming in Delphi

- Immutable variables - hard
- Pattern matching – if / case
- Higher-order functions – anonymous methods
- Recursion – plain old pascal
Anonymous methods

• Nameless methods
• Can be stored in a variable, field, passed as parameter ...

• Internally implemented as an interface
Anonymous methods

• Binding variable values
• Easy way to define and use methods
• Easy to parameterize using code

Hands-on
Inline versus anonymous

```
a := 2 * 3;
```

```
mul := Multiply();
a := mul(2, 3);
```

```
a := Multiply()(2, 3);
```
reverse :: [a] -> [a]
reverse [] = []
reverse (x:xs) = reverse xs ++ [x]
fibRecurrence first second = first : 
  fibRecurrence second (first + second)
fibonacci = fibRecurrence 0 1
main = print (fibonacci !! 10)
Questions?