Functional Programming for Delphi Developers

Nick Hodges

https://www.nickhodges.com
https://www.codingindelphi.com/blog

“The Failure of State”

Uncle Bob Martin

https://www.youtube.com/watch?v=7Zlp9rKHGD4

Changing State

- **It is hard to reason about the code** if you don't know for sure whether or not your data is changed.
- **It is hard to follow the flow** if you need to look not only at the method itself, but also at the methods it calls.
- If you are building a multithreaded application, following and debugging the code becomes even harder.

- Create Immutable types

(http://enterprisecraftsmanship.com/2015/03/02/functional-c-immutability/)

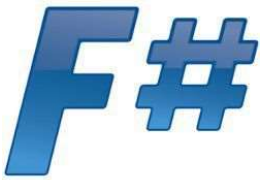
Immutable Types are good because…..

- With immutable class, **we need to validate its code contracts only once, in the constructor**.
- We absolutely sure that **objects are always in a correct state**.
- Objects are automatically **thread-safe**.
- **The code's readability is increased** as there's no need to step into the methods for making sure they don't change anything.

(http://enterprisecraftsmanship.com/2015/03/02/functional-c-immutability/)

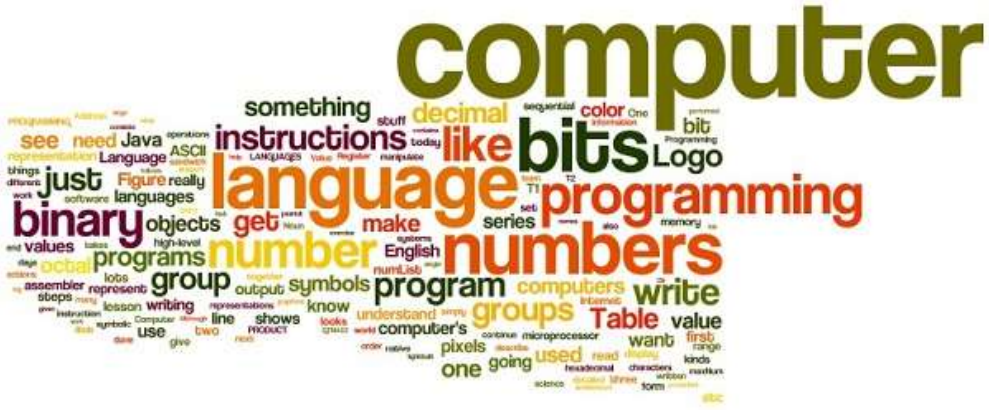
What is Functional Programming?

- A very hot topic
  - Haskell, Scala, Clojure, F#, Erlang, Groovy, etc.
- Lots of interesting claims
  - All the cool kids do it
  - Thread safety
  - Shorter, easier to write
  - Easier to test and debug



What is a Computer Program?

- First thing: What is a computer program?
  - A set of instructions telling the computer what to do
  - You tell the computer what to do, and it does it
- This way of doing/thinking is called “imperative” programming
- Imperative programming changes state
- HOW to achieve something

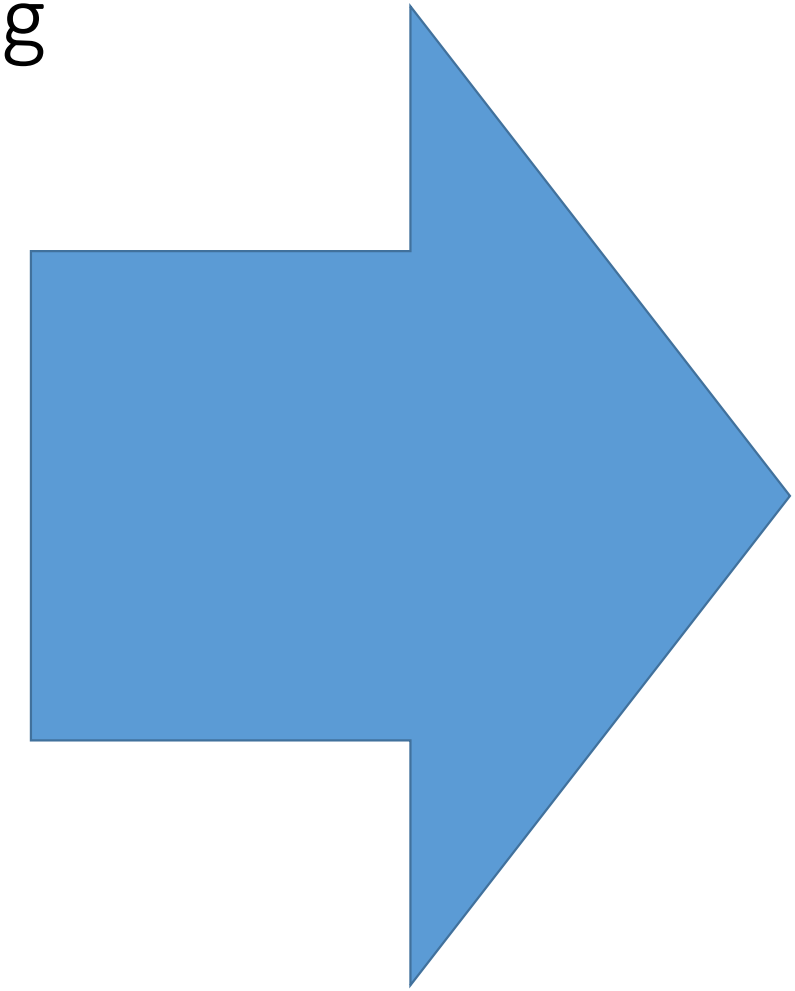


Imperative Programming

- Take the next company in the list
- If they owe money, send them a bill
- If there are more companies in the list, go back to the beginning

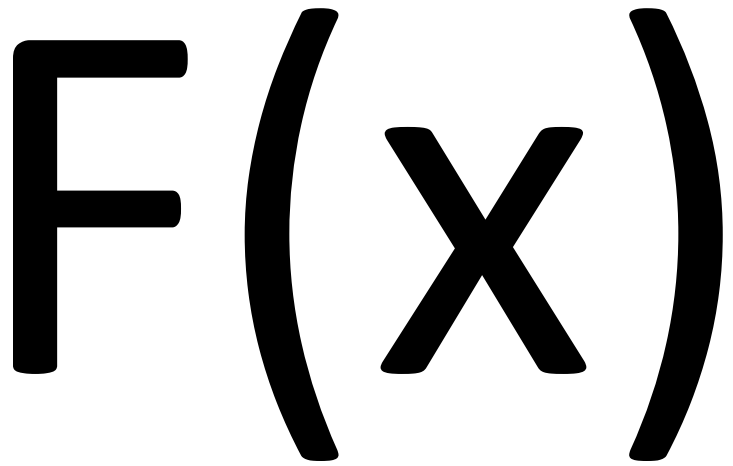
Functional Programming

- Send an invoice to all the customers in this list that owe us money



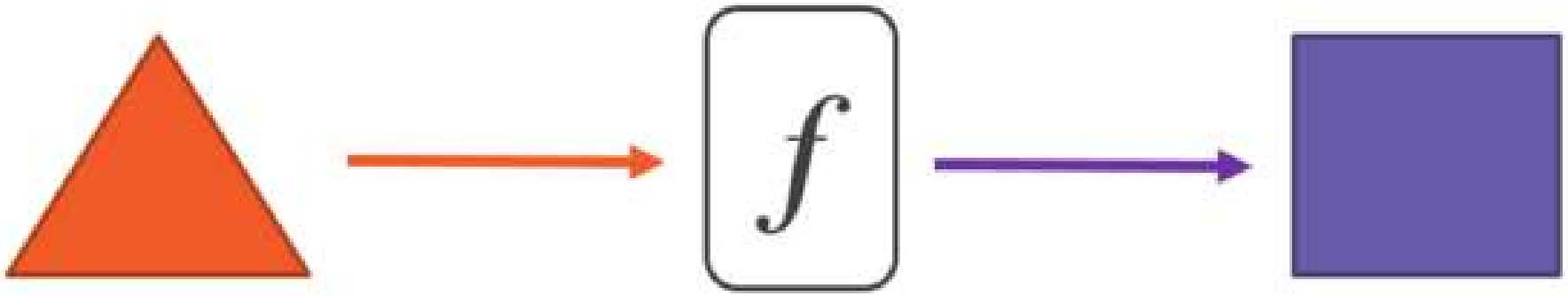
What is Functional Programming?

- Functional programming is different
  - a style of programming that emphasizes the evaluation of expressions rather than the ordered execution of instructions.
- It defines WHAT you want done
- Requires a radical mind shift



"OO makes code understandable by encapsulating moving parts. FP makes code understandable my minimizing moving parts"

Michael Feathers



- Referentially Transparent
- Method Signature Honesty

```
Assert.True(f(x), f(x));
```

For this to be true, the function must be “Referentially Transparent” and not change any state at all.

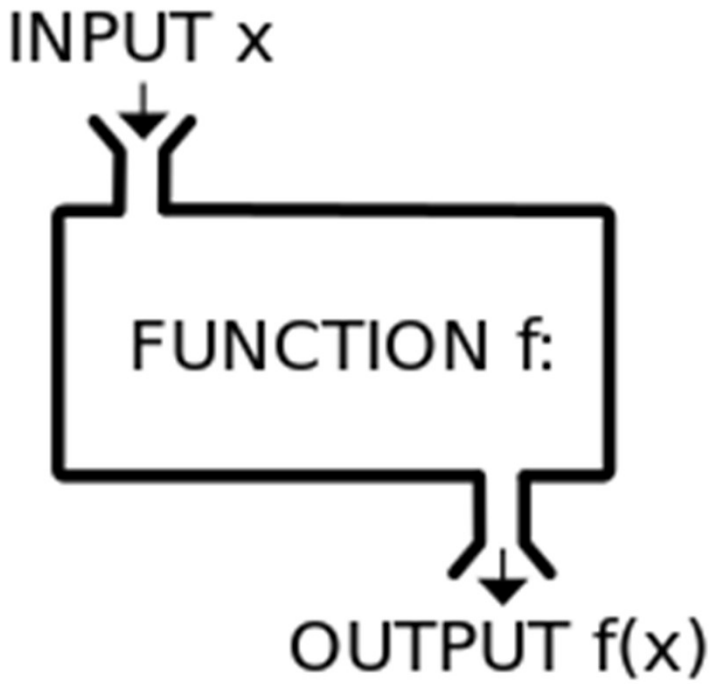
```
function GetElapsedTime(const aThen: TDateTime): TTimeSpan;
begin
  Result := Now - aThen;
end;


function sGetElapsedTime(const aThen, aNow: TDateTime): TTimeSpan;
begin
  Result := aNow - aThen;
end;
```

Features of functions

- Functions are:
  - The building blocks of a functional language
  - Encapsulated
    - They are a black box
  - Deterministic (“Pure”)
    - Always give the same result for a given input (Referential Transparency)
    - Idempotent
    - They never change state, no side effects, never refer to global state
  - Commutative
    - Doesn’t matter what order they are executed in

What are Functions?

- First class citizens
  - Basic unit of abstraction
  - They can be passed as parameters
  - They can be return values as other functions
  - You can replace the result with the function itself
  - Can appear anywhere other language constructs can appear
  - NO SIDE EFFECTS



Functional Programs are:

- Simpler (Fewer lines, no worrying about state of variable)
- Easier to write and maintain
- No temporal coupling or side effects
- Fewer to no concurrency issues
- No asking "What's the state"

Ways to Think

- Think about immutability over state transitions
- Think about results over steps (How would I do this if I couldn't iterate?)
- Think about composition over structure.
- Think about declarative over imperative

Or…..think of what you *can't* do

- You can't vary your variables
  - No assignment statements
- No looping
- Can't change your data structures
- Can't have any side effects

Remember when they told you "No GOTO statements"?

Of course…

- …there has to be some compromise.
- **`WriteLn`** is a side effect
- All functional languages have some compromises to interact with the user

Delphi Examples

- Display the functional way of thinking and writing code
- Uses IEnumerable<T> from Spring4D to make much of it happen
- Anonymous methods
- Try to ban the **`for`** loop

Remember…

- A **pure function** is a function which:
  - Given the same inputs, always returns the same output, and
  - Has no side-effects
- Functional Programming doesn’t like state change
  - An assignment operator changes state
- Functional Programming doesn’t like shared state

Questions ---

Buy my books at: http://www.codingindelphi.com

Delphi Books from Nick Hodges