What is declarative programming? [closed]

Asked 16 years, 3 months ago Modified 6 years, 8 months ago

Viewed 148k times

193

votes

Closed. This question needs to be more focused. It is not currently accepting answers.

Closed 9 years ago.

Locked. This question and its answers are locked because the question is off-topic but has historical significance. It is not currently accepting new answers or interactions.

I keep hearing this term tossed around in several different contexts. What is it?

programming-languages declarative glossary

Share

edited Sep 29, 2008 at 11:25

Sam Hasler

12.6k ● 10 ● 73 ● 106

asked Sep 24, 2008 at 20:15

Brian G
54.9k • 58 • 127 • 141

7 The answer you selected to be the correct one (indicated by the green check mark), is incorrect. It does not define what distinguishes declarative programming from its antithesis--imperative programming. Please consider changing your selection. – Shelby Moore III Dec 3, 2011 at 6:51

3 Yes, the answer marked as correct is NOT correct. – Dermot Feb 14, 2012 at 1:35

4 @ShelbyMooreIII Also specify which answer is correct so that we can read it! – vivek.m Jun 7, 2012 at 11:50

@vivek.m I have provided a new **answer** today.
– Shelby Moore III Jul 6, 2012 at 18:11

Comments disabled on deleted / locked posts / reviews

18 Answers

Sorted by: Highest score (default)

148
votes

Declarative programming is when you write your code in such a way that it describes what you want to do, and not how you want to do it. It is left up to the compiler to figure out the how.

Examples of declarative programming languages are SQL and Prolog.

Share

edited Aug 28, 2016 at 9:19

Aquarius_Girl
22.8k • 68 • 247 • 438

answered Sep 24, 2008 at 20:17

1800 INFORMATION
135k ● 30 ● 163 ● 242

29 You still to figure out "how" to tell the computer "what" you want :) – hasen May 14, 2009 at 21:14

7 @hasenj this and the other answers don't define the only attribute not shared with imperative programming-- which is **immutability**. – Shelby Moore III Dec 3, 2011 at 5:56

3 It will be really great if you can mention on how it is different from imperative programming (languages like C, C++, C#) then it will be more easier for readers to make out the difference. – RBT Dec 7, 2016 at 23:44

1 **programmer**: "I want to travel to Paris." **declarative(c)**: "How would you like to get there? sail by boat? or fly in an airplane? maybe, sail halfway and then fly the rest of the way there?" **programmer**: "I have no interest in how it's done." **imperative(sql)**: "Don't worry. I can query for what you need." (*this is how I understand the answer*) – nate Aug 21, 2017 at 7:05

How can SQL be declarative if it supports expressions that are not referentially transparent? – java-addict301 Sep 15, 2017 at 15:35

80 votes

The other answers already do a fantastic job explaining what declarative programming is, so I'm just going to provide some examples of why that might be useful.

Context Independence

Declarative Programs are *context-independent*. Because they only declare what the ultimate goal is, but not the intermediary steps to reach that goal, the same program can be used in different contexts. This is hard to do with *imperative programs*, because they often depend on the context (e.g. hidden state).

Take `yacc` as an example. It's a parser generator aka. compiler compiler, an external declarative DSL for describing the grammar of a language, so that a parser for that language can automatically be generated from the description. Because of its context independence, you can do many different things with such a grammar:

- Generate a C parser for that grammar (the original use case for `yacc`)
- Generate a C++ parser for that grammar
- Generate a Java parser for that grammar (using Jay)
- Generate a C# parser for that grammar (using GPPG)
- Generate a Ruby parser for that grammar (using Racc)
- Generate a tree visualization for that grammar (using GraphViz)
- simply do some pretty-printing, fancy-formatting and syntax highlighting of the yacc source file itself and include it in your Reference Manual as a syntactic specification of your language

And many more …

Optimization

Because you don't prescribe the computer which steps to take and in what order, it can rearrange your program much more freely, maybe even execute some tasks in parallel. A good example is a query planner and query optimizer for a SQL database. Most SQL databases allow you to display the query that they are *actually* executing vs. the query that you *asked* them to execute. Often, those queries look *nothing* like each other. The query planner takes things into account that you wouldn't even have dreamed of: rotational latency of the disk platter, for example or the fact that some completely different application for a completely different user just executed a similar query and the table that you are joining with and that you worked so hard to avoid loading is already in memory anyway.

There is an interesting trade-off here: the machine has to work harder to figure out *how* to do something than it would in an imperative language, but when it *does* figure it out, it has much more freedom and much more information for the optimization stage.

Share

answered Sep 28, 2008 at 3:35

Jörg W Mittag
369k ● 79 ● 453 ● 661

23

votes

Loosely:

Declarative programming tends towards:-

- Sets of declarations, or declarative statements, each of which has meaning (often in the problem domain) and may be understood independently and in isolation.

Imperative programming tends towards:-

- Sequences of commands, each of which perform some action; but which may or may not have meaning in the problem domain.

As a result, an imperative style helps the reader to understand the mechanics of what the system is actually doing, but may give little insight into the problem that it is intended to solve. On the other hand, a declarative style helps the reader to understand the problem domain and the approach that the system takes towards the solution of the problem, but is less informative on the matter of mechanics.

Real programs (even ones written in languages that favor the ends of the spectrum, such as ProLog or C) tend to have both styles present to various degrees at various points, to satisfy the varying complexities and communication needs of the piece. One style is not superior to the other; they just serve different purposes, and, as with many things in life, moderation is key.

Share

edited Dec 30, 2017 at 12:34

answered Jul 20, 2012 at 21:49

William Payne
3,285 3 25 25

That's the correct answer. Not any of that mumble above – Krzysztof Wende Dec 1, 2017 at 12:52

Thank you for not only answering the question, but doing it "Explain Like I'm 5" style with context and practicality. Excellent answer. – monsto Dec 15, 2019 at 16:02

17

votes

Here's an example.

In CSS (used to style HTML pages), if you want an image element to be 100 pixels high and 100 pixels wide, you simply "declare" that that's what you want as follows:

```
#myImageId {
height: 100px;
width: 100px;
}
```

You can consider CSS a declarative "style sheet" language.

The browser engine that reads and *interprets* this CSS is free to make the image appear this tall and this wide however it wants. Different browser engines (e.g., the engine for IE, the engine for Chrome) will implement this task differently.

Their unique implementations are, of course, NOT written in a declarative language but in a procedural one like Assembly, C, C++, Java, JavaScript, or Python. That code is a bunch of steps to be carried out step by step (and

might include function calls). It might do things like interpolate pixel values, and render on the screen.

Share

edited Jul 14, 2014 at 23:50

answered Jul 14, 2014 at 23:45

Niko Bellic

2,550 ● 2 ● 32 ● 26

11

votes

I am sorry, but I must disagree with many of the other answers. I would like to stop this muddled misunderstanding of the definition of declarative programming.

Definition

Referential transparency (RT) of the sub-expressions is the *only required* attribute of a declarative programming expression, because it is the only attribute which is not shared with imperative programming.

Other cited attributes of declarative programming, derive from this RT. Please click the hyperlink above for the detailed explanation.

Spreadsheet example

Two answers mentioned spreadsheet programming. In the cases where the spreadsheet programming (a.k.a. formulas) does not access mutable *global* state, then it is

declarative programming. This is because the mutable cell values are the monolithic *input* and *output* of the `main()` (the entire program). The new values are not written to the cells after each formula is executed, thus they are not mutable for the life of the declarative program (execution of all the formulas in the spreadsheet). Thus relative to each other, the formulas view these mutable cells as immutable. An RT function is allowed to access *immutable* global state (and also mutable *local* state).

Thus the ability to mutate the values in the cells when the program terminates (as an output from `main()`), does not make them mutable stored values in the context of the rules. The key distinction is the cell values are not updated after each spreadsheet formula is performed, thus the order of performing the formulas does not matter. The cell values are updated after all the declarative formulas have been performed.

Share

edited Apr 17, 2018 at 6:48

user8554766

answered Dec 3, 2011 at 5:48

Shelby Moore III

6,121 ● 1 ● 34 ● 36

1 Unintended side-effects can destroy the relationship between what was declared and the actual behavior of the program. I explained this in more detail in a new answer.
– Shelby Moore III Jun 22, 2013 at 16:44

8

votes

Declarative programming is the picture, where imperative programming is instructions for painting that picture.

You're writing in a declarative style if you're "Telling it what it is", rather than describing the steps the computer should take to get to where you want it.

When you use XML to mark-up data, you're using declarative programming because you're saying "This is a person, that is a birthday, and over there is a street address".

Some examples of where declarative and imperative programming get combined for greater effect:

- Windows Presentation Foundation uses declarative XML syntax to describe what a user interface looks like, and what the relationships (bindings) are between controls and underlying data structures.
- Structured configuration files use declarative syntax (as simple as "key=value" pairs) to identify what a string or value of data means.
- HTML marks up text with tags that describe what role each piece of text has in relation to the whole document.

Share

edited Sep 24, 2008 at 20:26

answered Sep 24, 2008 at 20:21

Chris Wenham
24k • 13 • 61 • 70

2 Although XML is declarative, I wouldn't go so far as to say it's declarative *programming* simply because there are no active semantics associated with the markup. Saying that something is an address does not help with figuring out what you want to do with it. – HenryR Sep 29, 2008 at 11:35

1 There has to be an underlying context (domain?) in which the declarative program is used. So using XML combined with ANT can be construed as a declarative program. – Gutzofter Sep 7, 2009 at 6:41

7 votes

Declarative Programming is programming with declarations, i.e. declarative sentences. Declarative sentences have a number of properties that distinguish them from imperative sentences. In particular, declarations are:

- commutative (can be reordered)
- associative (can be regrouped)
- idempotent (can repeat without change in meaning)
- monotonic (declarations don't subtract information)

A relevant point is that these are all structural properties and are orthogonal to subject matter. Declarative is not about *"What vs. How"*. We can declare (represent and constrain) a *"how"* just as easily as we declare a *"what"*. **Declarative is about structure, not content.** Declarative programming has a significant impact on how we abstract

and refactor our code, and how we modularize it into subprograms, but not so much on the domain model.

Often, we can convert from imperative to declarative by adding context. E.g. from "Turn left. (... wait for it ...) Turn Right." to "Bob will turn left at intersection of Foo and Bar at 11:01. Bob will turn right at the intersection of Bar and Baz at 11:06." Note that in the latter case the sentences are idempotent and commutative, whereas in the former case rearranging or repeating the sentences would severely change the meaning of the program.

Regarding *monotonic*, declarations can add *constraints* which subtract *possibilities*. But constraints still add information (more precisely, constraints are information). If we need time-varying declarations, it is typical to model this with explicit temporal semantics - e.g. from "the ball is flat" to "the ball is flat at time T". If we have two contradictory declarations, we have an inconsistent declarative system, though this might be resolved by introducing *soft* constraints (priorities, probabilities, etc.) or leveraging a paraconsistent logic.

answered Aug 6, 2012 at 21:43

dmbarbour

385 ● 4 ● 7

1 Declarative expressions contribute to the *intended* behavior of the program, imperative can contribute to intended or unintended. Declarative doesn't need to be commutative and idempotent, if this is intentional semantics. – Shelby Moore III Feb 28, 2013 at 7:54

6 votes

Describing to a computer what you want, not how to do something.

Share

answered Sep 24, 2008 at 21:57

denonde

99 ● 1

6 votes

imagine an excel page. With columns populated with formulas to calculate you tax return.

All the logic is done declared in the cells, the order of the calculation is by determine by formula itself rather than procedurally.

That is sort of what declarative programming is all about. You declare the problem space and the solution rather than the flow of the program.

Prolog is the only declarative language I've use. It requires a different kind of thinking but it's good to learn if just to expose you to something other than the typical procedural programming language.

Share

answered Sep 24, 2008 at 22:47

paan
7,182 • 8 • 40 • 44

6
votes

I have refined my understanding of declarative programming, since Dec 2011 when I provided an answer to this question. Here follows my current understanding.

The long version of my understanding (research) is detailed at **this link**, which you should read to gain a deep understanding of the summary I will provide below.

Imperative programming is where mutable state is stored and read, thus the ordering and/or duplication of program instructions can alter the behavior (semantics) of the program (and even cause a bug, i.e. unintended behavior).

In the most naive and extreme sense (which I asserted in my prior answer), declarative programming (DP) is avoiding all stored mutable state, thus the ordering and/or duplication of program instructions can **NOT** alter the behavior (semantics) of the program.

However, such an extreme definition would not be very useful in the real world, since nearly every program involves stored mutable state. The spreadsheet example conforms to this extreme definition of DP, because the entire program code is run to completion with one static copy of the input state, before the new states are stored. Then if any state is changed, this is repeated. But most real world programs can't be limited to such a monolithic model of state changes.

A more useful definition of DP is that the ordering and/or duplication of programming instructions do not alter any opaque semantics. In other words, there are not hidden random changes in semantics occurring-- any changes in program instruction order and/or duplication cause only intended and transparent changes to the program's behavior.

The next step would be to talk about which programming models or paradigms aid in DP, but that is not the question here.

Share

edited May 23, 2017 at 12:10

Community Bot

1 ● 1

answered Jul 6, 2012 at 18:08

Shelby Moore III

6,121 ● 1 ● 34 ● 36

Update: please refer also to the more exhaustive explanation at my other answer on the definition of declarative programming. – Shelby Moore III Nov 25, 2015 at 22:06

`Functional programming` is a buzz word these days which is essentially a subset of declarative programming. LINQ in C# language is an element of functional programming when the language itself is imperative by nature. So C# becomes sort of hybrid going by that definition. – RBT Dec 7, 2016 at 23:42

1 The compute.com link is dead. – Kedar Mhaswade Jul 23, 2017 at 13:14

5

votes

It's a method of programming based around describing *what* something should do or be instead of describing *how* it should work.

In other words, you don't write algorithms made of expressions, you just layout how you want things to be. Two good examples are HTML and WPF.

This Wikipedia article is a good overview: http://en.wikipedia.org/wiki/Declarative_programming

Share

answered Sep 24, 2008 at 20:20

Kevin Berridge

6,300 ● 6 ● 43 ● 44

1 Minor quibble. WPF is a library, not really a language or a paradigm. I think you really meant to say XAML is an example of declarative language. – Nick Oct 10, 2008 at 13:36

And how would you describe programming using a library/framework? – Gutzofter Sep 7, 2009 at 6:42

It is incorrect to state that declarative programming can't contain expressions. The key distinction is the expressions can't mutate stored values. – Shelby Moore III Dec 3, 2011 at 6:08

HTML is not a programming language – lud Jan 7, 2014 at 14:40

5

votes

Since I wrote my prior answer, I have formulated a new definition of the declarative property which is quoted below.

I have also defined imperative programming as the dual property.

This definition is superior to the one I provided in my prior answer, because it is succinct and it is more general. But it may be more difficult to grok, because the implication of the incompleteness theorems applicable to programming and life in general are difficult for humans to wrap their mind around.

The quoted explanation of the definition discusses the role *pure* functional programming plays in declarative programming.

> **Declarative vs. Imperative**
>
> The declarative property is weird, obtuse, and difficult to capture in a technically precise definition that remains general and not ambiguous, because it is a naive notion that we can declare the meaning (a.k.a semantics) of the program without incurring unintended side effects. There is an inherent tension between expression of meaning and avoidance of unintended effects, and this tension actually derives from the incompleteness theorems of programming and our universe.
>
> It is oversimplification, technically imprecise, and often ambiguous to define declarative as **“*what to do*”** and imperative as **“*how to do*”**. An

ambiguous case is the "*what*" is the "*how*" in a program that outputs a program— a compiler.

Evidently the unbounded recursion that makes a language Turing complete, is also analogously in the semantics— not only in the syntactical structure of evaluation (a.k.a. operational semantics). This is logically an example analogous to Gödel's theorem— "*any complete system of axioms is also inconsistent*". Ponder the contradictory weirdness of that quote! It is also an example that demonstrates how the expression of semantics does not have a provable bound, thus we can't prove[2] that a program (and analogously its semantics) halt a.k.a. the Halting theorem.

The incompleteness theorems derive from the fundamental nature of our universe, which as stated in the Second Law of Thermodynamics is "*the entropy* (a.k.a. the # of independent possibilities) *is trending to maximum forever*". The coding and design of a program is never finished— it's alive!— because it attempts to address a real world need, and the semantics of the real world are always changing and trending to more possibilities. Humans never stop discovering new things (including errors in programs ;-).

To precisely and technically capture this aforementioned desired notion within this weird universe that has no edge (ponder that! there is no

> “outside” of our universe), requires a terse but deceptively-not-simple definition which will sound incorrect until it is explained deeply.
>
> Definition:
>
> > ---
> >
> > The declarative property is where there can exist only one possible set of statements that can express each specific modular semantic.
> >
> > The imperative property3 is the dual, where semantics are inconsistent under composition and/or can be expressed with variations of sets of statements.
> >
> > ---
>
> This definition of declarative is distinctively *local* in semantic scope, meaning that it requires that a modular semantic maintain its consistent meaning regardless where and how it's instantiated and employed in *global* scope. Thus each declarative modular semantic should be intrinsically orthogonal to all possible others— and not an impossible (due to incompleteness theorems) *global* algorithm or model for witnessing consistency, which is also the point of “*More Is Not Always Better*” by Robert Harper, Professor of Computer Science at Carnegie Mellon University, one of the designers of Standard ML.

> Examples of these modular declarative semantics include category theory functors e.g. the `Applicative`, nominal typing, namespaces, named fields, and w.r.t. to operational level of semantics then pure functional programming.
>
> Thus well designed declarative languages can more clearly express meaning, albeit with some loss of generality in what can be expressed, yet a gain in what can be expressed with intrinsic consistency.
>
> An example of the aforementioned definition is the set of formulas in the cells of a spreadsheet program— which are not expected to give the same meaning when moved to different column and row cells, i.e. cell identifiers changed. The cell identifiers are part of and not superfluous to the intended meaning. So each spreadsheet result is unique w.r.t. to the cell identifiers in a set of formulas. The consistent modular semantic in this case is use of cell identifiers as the input and output of *pure* functions for cells formulas (see below).
>
> Hyper Text Markup Language a.k.a. HTML— the language for static web pages— is an example of a highly (but not perfectly3) declarative language that (at least before HTML 5) had no capability to express dynamic behavior. HTML is perhaps the easiest language to learn. For dynamic behavior,

> an imperative scripting language such as JavaScript was usually combined with HTML. HTML without JavaScript fits the declarative definition because each nominal type (i.e. the tags) maintains its consistent meaning under composition within the rules of the syntax.
>
> A competing definition for declarative is the commutative and idempotent properties of the semantic statements, i.e. that statements can be reordered and duplicated without changing the meaning. For example, statements assigning values to named fields can be reordered and duplicated without changed the meaning of the program, if those names are modular w.r.t. to any implied order. Names sometimes imply an order, e.g. cell identifiers include their column and row position— moving a total on spreadsheet changes its meaning. Otherwise, these properties implicitly require *global* consistency of semantics. It is generally impossible to design the semantics of statements so they remain consistent if randomly ordered or duplicated, because order and duplication are intrinsic to semantics. For example, the statements “Foo exists” (or construction) and “Foo does not exist” (and destruction). If one considers random inconsistency endemical of the intended semantics, then one accepts this definition as general enough for the declarative property. In essence this definition is vacuous as a generalized definition because it attempts to make

consistency orthogonal to semantics, i.e. to defy the fact that the universe of semantics is dynamically unbounded and can't be captured in a *global* coherence paradigm.

Requiring the commutative and idempotent properties for the (structural evaluation order of the) lower-level operational semantics converts operational semantics to a declarative *localized* modular semantic, e.g. **pure** functional programming (including recursion instead of imperative loops). Then the operational order of the implementation details do not impact (i.e. spread *globally* into) the consistency of the higher-level semantics. For example, the order of evaluation of (and theoretically also the duplication of) the spreadsheet formulas doesn't matter because the outputs are not copied to the inputs until after all outputs have been computed, i.e. analogous to pure functions.

C, Java, C++, C#, PHP, and JavaScript aren't particularly declarative. Copute's syntax and Python's syntax are more declaratively coupled to intended results, i.e. consistent syntactical semantics that eliminate the extraneous so one can readily comprehend code after they've forgotten it. Copute and Haskell enforce determinism of the operational semantics and encourage "don't repeat yourself" (DRY), because they only allow the pure functional paradigm.

2 Even where we can prove the semantics of a program, e.g. with the language Coq, this is limited to the semantics that are expressed in the typing, and typing can never capture all of the semantics of a program— not even for languages that are not Turing complete, e.g. with HTML+CSS it is possible to express inconsistent combinations which thus have undefined semantics.

3 Many explanations incorrectly claim that only imperative programming has syntactically ordered statements. I clarified this confusion between imperative and functional programming. For example, the order of HTML statements does not reduce the consistency of their meaning.

Edit: I posted the following comment to Robert Harper's blog:

> in functional programming ... the range of variation of a variable is a type

Depending on how one distinguishes functional from imperative programming, your ‘assignable’ in an imperative program also may have a type placing a bound on its variability.

> The only non-muddled definition I currently appreciate for functional programming is a) functions as first-class objects and types, b) preference for recursion over loops, and/or c) pure functions— i.e. those functions which do not impact the desired semantics of the program when memoized (thus perfectly pure functional programming doesn't exist in a general purpose denotational semantics due to impacts of operational semantics, e.g. memory allocation).
>
> The idempotent property of a pure function means the function call on its variables can be substituted by its value, which is not generally the case for the arguments of an imperative procedure. Pure functions seem to be declarative w.r.t. to the uncomposed state transitions between the input and result types.
>
> But the composition of pure functions does not maintain any such consistency, because it is possible to model a side-effect (global state) imperative process in a pure functional programming language, e.g. Haskell's IOMonad and moreover it is entirely impossible to prevent doing such in any Turing complete pure functional programming language.
>
> As I wrote in 2012 which seems to the similar consensus of comments in your recent blog, that declarative programming is an attempt to capture the notion that the intended semantics are never

> opaque. Examples of opaque semantics are dependence on order, dependence on erasure of higher-level semantics at the operational semantics layer (e.g. casts are not conversions and reified generics limit higher-level semantics), and dependence on variable values which can not be checked (proved correct) by the programming language.
>
> Thus I have concluded that only non-Turing complete languages can be declarative.
>
> Thus one unambiguous and distinct attribute of a declarative language could be that its output can be proven to obey some enumerable set of generative rules. For example, for any specific HTML program (ignoring differences in the ways interpreters diverge) that is not scripted (i.e. is not Turing complete) then its output variability can be enumerable. Or more succinctly an HTML program is a pure function of its variability. Ditto a spreadsheet program is a pure function of its input variables.
>
> So it seems to me that declarative languages are the antithesis of unbounded recursion, i.e. per Gödel's second incompleteness theorem self-referential theorems can't be proven.
>
> Lesie Lamport wrote a fairytale about how Euclid might have worked around Gödel's incompleteness theorems applied to math proofs

> in the programming language context by to congruence between types and logic (Curry-Howard correspondence, etc).

Share

edited May 23, 2017 at 10:31

Community Bot

1 ● 1

answered Mar 13, 2013 at 10:14

Shelby Moore III

6,121 ● 1 ● 34 ● 36

4

votes

Declarative programming is "the act of programming in languages that conform to the mental model of the developer rather than the operational model of the machine".

The difference between declarative and imperative programming is well illustrated by the problem of parsing structured data.

An imperative program would use mutually recursive functions to consume input and generate data. A declarative program would express a grammar that defines the structure of the data so that it can then be parsed.

The difference between these two approaches is that the declarative program creates a new language that is more closely mapped to the mental model of the problem than is its host language.

Share

edited Jul 20, 2012 at 20:57

answered Jul 20, 2012 at 20:49

dan_waterworth

6,431 ● 1 ● 33 ● 44

2

votes

It may sound odd, but I'd add Excel (or any spreadsheet really) to the list of declarative systems. A good example of this is given here.

Share

edited May 6, 2009 at 6:30

answered May 1, 2009 at 8:12

Lunatik

3,938 ● 6 ● 38 ● 53

1

vote

I'd explain it as DP is a way to express

- A *goal expression*, the conditions for - what we are searching for. Is there one, maybe or many?
- Some known facts
- Rules that extend the know facts

...and where there is a deduct engine usually working with a *unification* algorithm to find the goals.

Share

answered Sep 29, 2008 at 11:39

epatel
46k ● 17 ● 111 ● 144

-1
votes

As far as I can tell, it started being used to describe programming systems like Prolog, because prolog is (supposedly) about declaring things in an abstract way.

It increasingly means very little, as it has the definition given by the users above. It should be clear that there is a gulf between the declarative programming of Haskell, as against the declarative programming of HTML.

Share

answered Sep 24, 2008 at 22:40

Marcin
49.8k ● 18 ● 132 ● 206

1 There is no "gulf between the declarative programming of Haskell, as against the declarative programming of HTML", because the root attribute of declarative programming is immutability of stored values. – Shelby Moore III Dec 3, 2011 at 6:14

Be that as it may, there's a fair difference between a domain-specific language which is constrained in even its implicit calculations, as against a turing-complete programming system. – Marcin Dec 4, 2011 at 10:41

Agreed. Turing-completeness is orthogonal to immutability of stored values. So we should not conflate with the declarative vs. imperative attribute. Thanks for thinking for one of the attributes (Turing-completeness) that can cause that "gulf". – Shelby Moore III Dec 4, 2011 at 16:25

Turning-completeness only requires unbounded recursion. Immutability of stored values does not preclude unbounded

recursion. Thus they are orthogonal attributes.
– Shelby Moore III Dec 4, 2011 at 16:41

-2

votes

A couple other examples of declarative programming:

- ASP.Net markup for databinding. It just says "fill this grid with this source", for example, and leaves it to the system for how that happens.
- Linq expressions

Declarative programming is nice because it can help simplify your mental model* of code, and because it might eventually be more scalable.

For example, let's say you have a function that does something to each element in an array or list. Traditional code would look like this:

```
foreach (object item in MyList)
{
   DoSomething(item);
}
```

No big deal there. But what if you use the more-declarative syntax and instead define DoSomething() as an Action? Then you can say it this way:

```
MyList.ForEach(DoSometing);
```

This is, of course, more concise. But I'm sure you have more concerns than just saving two lines of code here and

there. Performance, for example. The old way, processing had to be done in sequence. What if the .ForEach() method had a way for you to signal that it could handle the processing in parallel, automatically? Now all of a sudden you've made your code multi-threaded in a very safe way and only changed one line of code. And, in fact, there's a an extension for .Net that lets you do just that.

- *If you follow that link, it takes you to a blog post by a friend of mine. The whole post is a little long, but you can scroll down to the heading titled* "The Problem" _and pick it up there no problem.*

Share

edited Apr 4, 2013 at 15:11

answered Sep 24, 2008 at 20:39

Joel Coehoorn

415k ● 114 ● 577 ● 813

1 You are describing functional programming, not declarative programming. Declarative programming has the attribute that it does not mutate stored values. – Shelby Moore III Dec 3, 2011 at 6:38

Declarative programming *can* mutate stored values... it's just that you specify (declare) *what* you want to mutate instead of exactly how exactly to go about mutating it. What else do you think an sql INSERT or UPDATE statement in sql does? – Joel Coehoorn Apr 4, 2013 at 15:13

You are missing the point that if your functions are not pure, then unintended side-effects can destroy the relationship between what you declared and the actual behavior of the

program. I explained this in more detail in a new answer.
– Shelby Moore III Jun 22, 2013 at 16:41

-3

votes

It depends on how you submit the answer to the text. Overall you can look at the programme at a certain view but it depends what angle you look at the problem. I will get you started with the programme: Dim Bus, Car, Time, Height As Integr

Again it depends on what the problem is an overall. You might have to shorten it due to the programme. Hope this helps and need the feedback if it does not. Thank You.

Share

answered Jun 30, 2014 at 10:37

Danny Darrie

1