A polyglot day: learning from language paradigms

Benson Joeris

Kathleen Dollard

What if you understood all the paradigms that define modern languages?

You could program better in your base language,
choose the best tool for the job,
integrate ideas from other languages and
design systems that cross boundaries from legacy to the future.

Awesome!

But, who has time to do the hard work of learning a bunch of languages from the inside out to be a true polyglot programmer?

Benson Joeris

Theoretical mathematician who loves languages

- Post-doctoral researcher, University of Waterloo
- Expert in algorithms, functional programming and type systems who's worked and experimented with languages across all of today's paradigms and thinks about problems of dependability, performance and scale from a perspective of core principles
- Pluralsight
  - Haskell Fundamentals

Kathleen Dollard

Passionate pragmatic programmer

- Decades in industry
- Director of Engineering, ROI Code, Denver
  - Love leading teams developing business applications
- Microsoft MVP since 1998
- Teacher, speaker, writer
  - Pluralsight
    - Event Tracing for Windows (ETW) in .NET
    - .NET Puzzles
    - What's New in .NET 4.5
    - Visual Studio 2015 Analyzers
    - Understanding Metaprogramming
  - WilltellectNOW
    - C# 6

Schedule

- Section 1
  - Hello World in 4 languages
- Section 2
  - Language paradigms
- Section 3
  - Advanced paradigms
- Section 4
  - Pulling together languages, libraries and tools

Goals

- Understanding how languages are different and most have value
  - And the realization of how many characteristics are converging
- What can you learn from each language you can take back to your core language
- How to choose the best language for a problem/project

Section 1: Basics

History

Hello World in Each Language

Language History

- FORTRAN (1954)
  - John Backus, IBM, early high level general purpose language
- BASIC (1964)
  - Kemeny and Kurtz, Dartmouth, Beginner's All-purpose Symbolic Instruction Code
- C (1972)
  - C: Dennis Ritchie developed for Unix based on B (Martin Richards, Ken Thompson)
- C++ (1983)
  - C++: Bjarne Stroustrup set out to enhance the C language with Simula-like features
- Haskell (1990)
  - Built by committee to consolidate the existing functional languages for language research
- Python (1994 (v1))
  - Guido van Rossum over 6 years initially as a hobby. Declared Benevolent Dictator for Life
- Javascript (1995)
  - Brendan Eich originally developed it in 10 days, while working for Netscape
- C# (2000)
  - Anders Hejlsberg and a team at Microsoft resources built a language for widespread use
- F# (2005)
  - Don Syme, Microsoft Research, functional programming in the .NET Framework

From Wikipedia.org

- ***C#***, Java, Visual Basic (strict on), Objective C
  - Object oriented, strongly typed, functional features
- ***Python***
  - Object oriented, dynamically typed, functional features
- ***Haskell***
  - Non-OO, strongly typed (different from C#, Java), functional
- ***JavaScript***
  - Unstructured objects, dynamically typed, functional features
- ***F#*** (ML – Caml), Scala
  - Crossover language
  - OO features, strongly typed (different from C#, Java), functional
- C++
  - Object oriented (multiple inheritance), strongly typed, functional features, pointers

- Agda , Idris
  - Dependently typed
- Bash, PowerShell, Batch (DOS), Groovy, Lua
- Lisp, Clojure, Scheme
- TypeScript, CoffeeScript, Dart, TCL
- Erlang
  - Message/actor
- Ruby
- Perl
- Go, Eiffel, Kotlin, Logo, Nemerle, Processing, Rust, Scratch, Swift, Wolfram Language
- PHP
- PowerBuilder (4GL)
- ProLog

DSL

- MatLab, Mathematica, Magma, Sage, Octave
- R
- Geurkin/Specflow/Cucumber
- LaTeX, XML, XSLT, HTML, PostScript
- SQL variations

Others

- Hope, Joy, Io, Kaleidoscope, Linoleum, Pizza, Snowball, Squeak, Squirrel

What you do with this knowledge

- Find a better fit for particular problems
- Apply functional techniques to imperative, object-oriented languages
- Apply strongly typed ideas to dynamic languages
- See alternate methods of extension
- Check the performance of algorithms created in several different ways

Type systems

- A type system determines if data is compatible with an operation
  - What kind of data can be assigned to a variable, or passed to a function
- Functions have types, as well as their arguments/return values
  - “A function that takes an integer argument and returns an integer” is a type
- Types also help the compiler generate efficient machine code
  - But we will focus on how types help programmers, rather than how they help the compiler

DEMO

Hello Planets

Filtered aggregation in each language

Summary of Hello Planets

- Avoid being terrorized by syntax
- Dynamic vs. static typing
- Functions as first class types
- Loops and conditionals vs maps and filtering
- Ability to use similar approaches in different languages

DEMO

Crazy constructs and some helpful ones

Section 2: Language paradigms

Paradigm overview

Language paradigms

- Imperative
- Procedural
- Structured
- Object Oriented
- Functional
- Declarative

- Typing, equality and truth
- Static/dynamic typing
- Type inference
- Pattern matching/deconstruction
- Eager/lazy evaluation
- Purity vs. side effects
- Dependence on ecosystem
- Human readability
- Resilience to change

DEMO

Tax Calculator

Summary of Tax Calculator

- Big Picture
  - Fundamentally different approaches possible with different paradigms
- Details
  - Pattern matching
  - Classes and objects
    - Ability to add attributes in Python and Javascript
  - Public/private visibility
    - Python underscores
    - Javascript nested function scope
  - Interfaces
  - Function signatures
    - loose arguments (args, kwargs)

Section 3: Advanced paradigms

Function purity, side effects, and the real world

Async

DEMO

Lessons from Haskell: function purity, side effects and the real world

DEMO

Async

Section 4: Pulling it all together

Fitting languages to your problems

Multiple paradigms, implementations/platforms, language evolution and editors

Choosing a paradigm for your problem

- A project often involves different problems requiring different paradigms
- Languages support multiple paradigms
- Choosing a paradigm
  - Functional
    - Compute values from data
  - Imperative
    - Handling state
  - Object-oriented
    - Hierarchies of specialization and inheritance

Choosing a paradigm for your problem

- Complex calculations
  - Functional
- Simulations and games
  - Imperative/Object Oriented
- Basic CRUD
  - Anything, depends on library/ORM (maybe not pure functional)
- Big data
  - Libraries (SciPi)
- Basic web app
  - Depends on library
- Front end web
  - Many languages transpile to Javascript
- Front end native
  - Library dependent – C#, C++, Java?

Choosing a language for a project

- Library support
- Language strengths
  - Relevant paradigms
  - Resilience to change
- Multiple languages possible
  - Added complexity
  - Performance consideration

DEMO

F#

DEMO

Editors, PEP 8 and Linters

DEMO

ECMA 6

Zen of Python

- Beautiful is better than ugly.
- Explicit is better than implicit.
- Simple is better than complex.
- Complex is better than complicated.
- Flat is better than nested.
- Sparse is better than dense.
- Readability counts.
- Special cases aren't special enough to break the rules.
- Although practicality beats purity.
- Errors should never pass silently.
- Unless explicitly silenced.

https://www.python.org/dev/peps/pep-0020/

Zen of Python

- In the face of ambiguity, refuse the temptation to guess.
- There should be one— and preferably only one —obvious way to do it.
- Although that way may not be obvious at first unless you're Dutch.
- Now is better than never.
- Although never is often better than **right** now.
- If the implementation is hard to explain, it's a bad idea.
- If the implementation is easy to explain, it may be a good idea.
- Namespaces are one honking great idea—let's do more of those!

https://www.python.org/dev/peps/pep-0020/

THANK YOU!

Kathleen Dollard

kathleen@mvps.org

@kathleendollard on Twitter

Benson Joeris

bjoeris@uwaterloo.ca

@bjoeris on Twitter