1. Background
   1. History
      1. Several key events began to occur that sparked the movement for non-strict or “lazy” evaluation
         1. “Cons should not evaluate its arguments” by Dan Friedman and David Wise, and “A lazy evaluator” by Peter Henderson and James H. Morris Jr. were published in 1976.
         2. David Turner changed SASL to a non-strict language in 1976, and showed the usefulness of list processing by lazy evaluation.
         3. Some individuals were attempting to apply graph reduction principles to software.
         4. Research was conducted on the use of dataflow and graph reduction machines to solve problems as a non-von Neumann architecture.
         5. Over the next several years, several conferences and meet-ups began to emerge such as the first Lisp conference in August 1980, the Advanced Course on Functional Programming and its Applications in July 1981, the first conference on Functional Programming Languages and Computer Architecture (FPCA) in September 1981, and the second Lisp conference in September 1982 that marked the beginning of the biannual Lisp and Functional Programming conference (LFP).
         6. Many languages like Miranda, Lazy ML, Orwell, and Clean were developed to meet this growing interest in non-strict evaluation, but many seemed to implement similar features and none had much backing as Miranda, developed by David Turner.
      2. Haskell's genesis started with Simon Peyton Jones' visit to Paul Hudak at Yale on his trip to FPCA in the fall of 1987, upon which they “decided to initiate a meeting during FPCA, to garner interest in designing a new, common functional language” (p. 12-3), which was also encouraged by Philip Wadler.
         1. At the meeting it was decided that the best place to start developing the new language would be from an existing one. Since Miranda was deemed to be the best language for the task, the committee decided to first approach Turner about the adoption of his language. In response the request, Turner stated that he wished to prevent multiple dialects of Miranda and declined the committee invitation (Hudak et al., 2007).
         2. From this point, the committee decided to set up an email mailing list fplangc@cs.ucl.ac.uk in order to communicate remotely, and eventually met up for their first planned meeting at Yale between January 9-12 in 1988. At this meeting, the group initially decided to establish its goals for the language as following:
            1. It should be suitable for teaching, research, and applications, including building large systems.
            2. It should be completely described via the publication of a formal syntax and semantics.
            3. It should be freely available. Anyone should be permitted to implement the language and distribute it to whomever they please.
            4. It should be usable as a basis for further language research.
            5. It should be based on ideas that enjoy a wide consensus.
            6. It should reduce unnecessary diversity in functional programming languages. OL became the language chosen as a baseline.

Ultimately, the group abandoned OL and exclusive use of proven ideas, and never developed formal semantics. The other main items of discussion were concerned with the committee process and the name of the language. At first, the group decided upon “Curry,” named after Haskell B. Curry, whose contribution to Combinatory Logic was the basis upon which functional programming was built; however the group decided to avoid confusion and puns by instead using “Haskell” with permission from Mrs. Curry.

* + 1. The next most important meeting was at the University of Glasgow, April 6-9, 1988, where many unresolved issues were discussed and Hudak and Wadler were designated as the editors for the first Haskell report. This was followed by other meetings and emails until the Haskell version 1.0 Report was published in April 1990, soon after which the original mailing list was disbanded in place of a public mailing list. Other subsequent reports included:
       1. Haskell version 1.1 Report (August 1991) – Let expressions and operator sections allowed for the first time.
       2. Haskell version 1.2 Report (March 1992) – Minor changes and an appearance in *SIGPLAN Notices*.
       3. Haskell version 1.3 Report (May 1996) – A library report was added to enhance portability, Monadic I/O was added and I/O semantics were dropped, type classes were generalized to higher kinds using constructor classes, and algebraic data types were extended with new-types, strictness annotations, and named fields.
       4. Haskell version 1.4 Report (April 1997) – List comprehensions were generalized to arbitrary monads.
       5. Haskell 98 Report (February 1999) – The community agreed to support of a stable standard for the language and list comprehensions reverted to just lists.
       6. Revised Haskell 98 Report (December 2002) – Cambridge University Press published the Report as a book while still allowing the entire text to be freely available online.

Several other significant events occurred during this time as well, such as the founding of Haskell's website, haskell.org, in 1994 which is maintained by Hudak's group at Yale, and the Haskell committee turned over control of the language to the Haskell community while setting Jones over the Report as the sole editor between 1999-2002.

* + 1. Regardless of Turner's decision to prevent dialects of Miranda, Haskell's development was highly influenced by it, which also makes Haskell a descendant of ML (Sebesta, 2004). Aspects of Miranda that influenced Haskell include the general methods of purity, higher, order, laziness, and static typing. Other similarities are found in terms of syntax for “the equational style of function definitions, especially pattern matching, guards, and where clauses; algebraic types; the notation for lists and list comprehensions; writing pair types as (num,bool) rather than the int\*bool of ML; capitalisation of data constructors; lexically distinguished user-defined infix operators; the use of a layout rule; and the naming of many standard functions” (Hudak et al., 2007).
  1. Contributions to the programming landscape (Mention this in Type classes and Monads)

1. General Facts
   1. Lazy evaluation
   2. Type system
   3. Purity?
   4. Syntax? (maybe best left to each section)
   5. Modules
      1. Prelude
2. Types
   1. Basics
   2. Declarations
3. Functions (subprograms)
   1. Purity?
   2. Control structures?
   3. Recursion?
4. Abstraction and Polymorphism
   1. Abstracting types
5. Type classes
   1. Basics
   2. Declarations
      1. Recursion?
6. I/O
7. Exception Handling
8. Monads
9. Concurrency
10. Sample Program
11. Global Issues and Promise for the Future
    1. As Haskell is approaching the 25th year since its initial publishing, it is important to stop and take into account its progress in order to correctly assess its promise for the future. In the beginning, Haskell began as an academically focused language with a relatively small following, but has grown from under 100 consistent users in 1995 to about 600 users in 2005. Those who have been impacted by Haskell include both academic and industrial groups. According to a web survey taken by Hudak and his associates (2007) for the academic year 2005-2006, 126 teachers from 89 universities among 22 countries; the highest responses being from the “USA (22%), the UK (19%), Germany (11%), Sweden (8%), Australia (7%), and Portugal (5%)”; were estimated to teach Haskell to a range of 5000-10,000 students (p. 12-41). The largest group of students taking these courses in Haskell was estimated to be 2000-4000 annually for undergraduates learning Haskell as their first or second language. The types of courses offering Haskell included a focus on functional or declarative programming, advanced programming, programming languages, theoretical topics, hardware descriptions, domain-specific languages, music, quantum computing, and distributed and parallel programming (Hudak et al., 2007).
    2. As for industrial uses, companies from around the world have decided to implement Haskell in order to enhance their products. Some of these companies and their respective countries include ABM AMRO, an international bank in Amsterdam, The Netherlands; Aetion Technnologies LLC, a defense contractor that conducted business between 1999-2011 in Columbus, Ohio with a focus on projects involving Artificial Intelligence; Better (or Erudify), founded in 2012 and based in both Zurich, Switzerland and New York, USA, used Haskell for its back-end web-servers and learning logic in order to provide high-quality courses; and bCODE Pty Ltd in Sydney, Australia; Fractis Research, which develops mobile-friendly solutions in Freiburg, and Germany; and Functor AB in Stockholm, Sweden, a company that produces tools to help eliminate bugs (Rheno, 2014). In addition, a man by the name of Curt Sampson attempted to outline his experiences with the transition to using Haskell as a functional language upon founding Starling Software in Tokyo, Japan. After reviewing the benefits of functional languages, Sampson and his team convinced their client to allow the use of Haskell. With the team's previous experience in the imperative languages of C, Java, Ruby, C# and Python, Sampson concluded that using Haskell had the advantages of concise and readable code, no noticeable disadvantages in speed thanks to concurrency and parallelism, portability, and the ability to interface with foreign functions in C. However, Sampson also mentions that problems did arise when dealing with refactoring small bits of code, a lack of profiling tools, and space/memory leaks that arose from use of threads, but then later states that some of these issues could likely be resolved with more experience.
    3. So, is Haskell projected to continue living? According to these developments in the academic and industrial communities, it seems to likely be so. This assessment may be further confirmed when evaluating the the active state of its community both in the IRC channel and the Haskell Wiki, which is the official Haskell website. For instance, the Haskell 2010 Report was just recently released, setting revised standards for the language by those who contributed to the mailing list and discussion boards. Considering all these factors, it may even be possible that an increase in developments may be seen in the near future.