The Origins of LISP, Scheme, and Functional Programming

- In the late 19th Century some people worried about the foundations of Mathematics. Russell’s Paradox: “Let S be the set of all sets that don’t contain themselves. Is S a member of S?” is neither True nor False. There were various attempts to formalize Mathematical language to prevent such a paradox from being expressed. Russell and Whitehead wrote *Principia Mathematica* – 3 volumes, with “1+1=2” proven halfway through volume 2. Russell required a set to have a type for its elements. Since the type of the set is “set of type x” and all the members would have type “x”, no set could be a member of itself. Other attempts to fix the problem included Combinatory Logic (Moses Schonfield) which replaced quantifiers with functions and Recursive Function Theory, which studied computability in terms of mathematical notions of functions. All of these attempts to avoid paradoxes were made pointless in 1931 by Godel’s Incompleteness Theorem, which showed that in any mathematical system rich enough to include basic arithmetic, propositions could be expressed that could neither be proven nor disproven.

- In 1928 David Hilbert posed the Entscheidungsproblem which was to show that all “Decision Problems” (questions with yes/no answers) could be solved algorithmically. In 1936 Alonzo Church (at Princeton) invented the lambda calculus to show this was false. Later that year Alan Turing invented Turing Machines to show again that the Entscheidungsproblem was false. After WWII, as interest in computation grew, there were numerous models of computability including
  - Turing Machines
  - Church’s lambda-calculus
  - Recursive Function theory
and others. These were all shown to be equivalent. Church’s Thesis (first formally stated by Stephen Kleene in 1943) says that Turing Machines (hence all of the others) embody our informal notion of what it means to be an algorithm.

- John McCarthy (at MIT in the mid 1950s) was interested in both AI and questions of computability. Much of the AI work at this time involved complex association lists, which FORTRAN did not process easily. McCarthy described LISP in 1958 as a formal language for describing AI algorithms. He used the lambda calculus so he could make use of Church’s work to show that LISP was Turing-complete. McCarthy did not intend to implement LISP; it was just a language for describing algorithms. Steve Russell (MIT) implemented LISP in 1960. (Russell later wrote *Space War*, the first computer game.)

- In the mid 1970’s Guy Steele was a graduate student at MIT, working under Gerald Jay Sussman. They wrote a series of papers about the implementation of LISP, including why is should be lexically rather than dynamically scoped and how tail recursions could be turned into loops. They were also interested in the idea of a continuation from Programming Language Theory (a formalization of control structures) Continuations were used in the LISP implementation but could not be manipulated by the programmer; Steele and Sussman thought they should have the same status as other data. These papers laid the foundations for a new language (Scheme) that was implemented in stages in the mid-to-late 1970s.