Introduction This paper is an attempt to demonstrate to the larger community of (nonfunctional) programmers the significance of functional programming, and also to help functional programmers exploit its advantages to the full by making it clear what those advantages are.It is a logical impossibility to make a language more powerful by omitting features, no matter how bad they may be. Even a functional programmer should be dissatisfied with these so-called advantages, because they give no help in exploiting the power of functional languages.To those more interested in material benefits, these "advantages" are totally unconvincing. Functional programmers argue that there are great material benefits -- that a functional programmer is an order of magnitude more productive than his or her conventional counterpart, because functional programs are an order of magnitude shorter. Yet why should this be? The only faintly plausible reason one can suggest on the basis of these "advantages" is that conventional programs consist of 90% assignment statements, and in functional programs these can be omitted!Since expressions can be evaluated at any time, one can freely replace variables by their values and vice versa -- that is, programs are "referentially transparent". This freedom helps make functional programs more tractable mathematically than their conventional counterparts. Such a catalogue of "advantages" is all very well, but one must not be surprised if outsiders don't take it too seriously.It says a lot about what functional programming isn't (it has no assignment, no side effects, no flow of control) but not much about what it is. The functional programmer sounds rather like a mediaeval monk, denying himself the pleasures of life in the hope that it will make him virtuous.There is no yardstick of program quality here, and therefore no ideal to aim at. Clearly this characterization of functional programming is inadequate.The special characteristics and advantages of functional programming are often summed up more or less as follows.Functional programming is so called because its fundamental operation is the application of functions to arguments.Functional programs contain no assignment statements, so variables, once given a value, never change.This eliminates a major source of bugs, and also makes the order of execution irrelevant -- since no sideeffect can change an expression's value, it can be evaluated at any time.If omitting assignment statements brought such enormous benefits then Fortran programmers would have been doing it for twenty years.We must find something to put in its place -- something that not only explains the power of functional programming but also gives a clear indication of what the functional programmer should strive towards.A main program itself is written as a function that receives the program's input as its argument and delivers the program's output as its result.Typically the main function is defined in terms of other functions, which in turn are defined in terms of still more functions, until at the bottom level the functions are language primitives.All of these functions are much like ordinary mathematical functions, and in this paper they will be defined by ordinary equations.We are following Turner's language Miranda[4]2 here, but the notation should be readable without specific knowledge of this.More generally, functional programs contain no side-effects at all.One cannot write a program that is particularly lacking in assignment statements, or particularly referentially transparent.This relieves the programmer of the burden of prescribing the flow of control.A function call can have no effect other than to compute its result.This is plainly ridiculous.