Functional programming

- **Building Abstractions with Functions**
    - Programming language provides for combining simple ideas to form more complex ideas.
        - **primitive expressions,** which represent the simplest entities the language is concerned with
        - **means of combination,** by which compound elements are built from simpler ones
        - **means of abstraction**, by which compound elements can be named and manipulated as units
- We have, in every language, at least two kind of elements for this:
    - primitive functions and primitive data (in the final analysis they are same)
    - methods of combining and abstracting functions and data
-
- **Expressions**
    - A number is an expression
    - (expression) is an expression
    - Combination is a expression -> expression operator expression -> 127 + 234;
    - Combinations can be nested -> (124 + 234) * 123
    - interpreter evaluates expressions in **read-eval-print-loop(REPL)**
    - Expression evaluation always results in a value
    - programming language provides for using names to refer to computation objects. Expression evaluation values can be given names -> let result = (124 + 234) * 123
    - Interpreter must maintain pairs of name and value. The memory which keep track name-value pairs is called an *environment*
- **Evaluating Operator combinations**
    - Evaluate the operand expressions
    - Apply the function that is denoted by the operators to the arguments that value of the operands
        - Notice evaluation is inherently recursive in nature
        - the values of the numerals are the numbers they name
        - the values of the names are the objects associated with those names in the environment

- **Functions**
    - compound operation can be given name and then referred to as a unit later
    - Similar to operator combinations (In fact, both are same)
- **The substitution model of function application**
    - to evaluate an application combination of the form :
        - function-expression (argument-expressions)
    - do the following:
        - Evaluate the function expression of the application combination, resulting in the function to be applied
        - Evaluate the argument expressions of the combination
        - Apply the function to the arguments:
            - If the function is primitive, we simply apply the corresponding mathematical function to the argument
            - If the function is compound, we evaluate the return expression of the function with each parameter replaced by corresponding argument
        - This is called applicative order evaluation model
    - There is an equivalent evaluation model called normal order
        - fully expand and reduce (would not evaluate operands until they are absolutely needed)