316 12-03-09

- Lisp:
  - Lisp has 2 major dialects:
    - Common Lisp
    - Scheme
  - We will look at pure Scheme, the functional sublanguage of Scheme – as an example of functional language
  - Pure functional languages – all computation is based on expression evaluation, chain of function calls
    - No loops, no assignments
    - Need to use recursion for all iterations
    - Pro:
      - Language can be learned faster
      - Programs can be written faster
    - Con:
      - Efficiency – slower execution
  - Symbolic Expressions (S_Expressions)
    - Main data structure
    - Lisp programs themselves are written in S_Expressions
    - BNF:
      - <S_Expression> → <atom> | "(" <S_Expression> "." <S_Expression> ")"
        {There must be at least one blank before/after the period}
      - <atom> → any string of printable characters except for blanks, " (", ") ", " ' "
    - Conservative (Cons) Expression:
      - "(" <S_Expression> "." <S_Expression> ")"
        - The first <S_Expression> is known as the car part
        - The second <S_Expression> is known as the cdr part
    - Special atom "()", called nil, denotes the empty S_Expression
    - <atom> includes numbers (like integers, floats, etc)
      - (CS316 . ABC)
      - (CS?#B . 123)
      - (CS316 . (B#5.7 . 58E-7))
      - Can be nested to any depth
        - ((2 . 5) . ((ABC . CS316) . (B# . 58E07)))
    - Cons Expressions are implemented in the heap
    - Implementation-wise, Cons Expressions are binary trees
  - The S_Expressions of the form (e_1 . (e_2 . … (e_n . ()) …)) are regarded as lists of *n* elements e_1, e_2, … , e_n [and abbreviated as (e_1 e_2 e_n), and there's at least one blank to separate e_i]
    - Each e_i may be any S_Expression/list nested to any depth
  - Example:
    - (1 2 3 4) = (1 . (2 . (3 . (4 . ( )))))
    - (1 (A B) 3 (C D)) = (1 . ((A . (B . ( ))) . (3 . ((C . (D . ( ))) . ( )))))
      - (AB) = (A . (B . ( )))
      - (CD) = (C . (D . ( )))

        - All lisp operator/function applications take the form (f e_1 e_2 … e_n)
            - Where f = operator/function
            - e_1 e_2 … e_n = arguments
    - Example: (f (a 1 2 3)) [f and a are functions]
        - (f '(a 1 2 3)) – applying f to the list of 4 elements; a, 1, 2, 3
            - [f is a function, a is an atom]
        - Apply ' to S_Expressions/lists when they are used as data arguments to functions
        - (car '(e_1 . e_2)) ➔ e_1
        - (cdr '(e_1 . e_2)) ➔ e_2
        - (cons 'e_1 'e_2) ➔ (e_1 . e_2)
    - (car '((1 2) . (3 4))) ➔ (1 2)
    - (cdr '((1 2) . (3 4))) ➔ (3 4)
    - (cons '(1 2) '(3 4)) ➔ ((1 2) . (3 4))
    - (car '(e_1 e_2 … e_n)) ➔ e_1 [the head of the list]
    - (cdr '(e_1 e_1 … e_n)) ➔ (e_2 … e_n) [the tail list]
    - (cons 'e_1 '(e_2 … e_n)) ➔ (e_1 e_2 … e_n)
    -