1. **Arrays** contain elements and provide either a **Reference (pointer)** or **Dereference (value)**
   A. Subscripts begin at 0: \( y = \text{array}[0] \);
   B. Array by subscript **derefneses** an element and yields a **Value**: \( y = \text{array}[3] \);
   C. Array by name (no subscript) yields a **Reference (pointer)**: \( py = \text{array} \)

2. **Functions:**
   - When a function is called, the **Parent** calls the **Child** function
   - When a function is called, only numbers are sent as parameters
   - Functions called with **value** parameters send a **copy of the value** to the function.
   - Functions called with a **reference** parameter send a **pointer (address)** to the function.

   **A. Declaring the function Prototype** - ex: int getval(int copy, int* ptr);<
   1. **Return value** data type is specified for function of given **Name**
   2. **Parameter declaration** by **Value**: a variable name
      - int foo( int x ); // copy of integer variable
   3. **Parameter declaration** by **Reference (pointer)**: array name or variable pointer
      - int foo( int* apples); // integer pointer
      - int foo( int apples[] ); // integer pointer (braces indicate array declaration)
      - int foo( int apples[25] ); // integer pointer (25 is ignored - in a declaration)
   - **......**
      - int foo( char* str ); //called with a character pointer
      - int foo( char str[] ); //called with a character pointer
   4. **Using the function (calling it)** - within the application code
      1. **By Value**: variable
         - y = foo(x); // call function with a copy of value of integer x
      2. **By Reference (pointer)**: array
         - 1. status = foo( str ); // call with pointer to first char of str array (string)
         - 2. status = foo( apples ); // call with pointer to integer array apples (above declaration)
         - 3. \( y = \text{foo}(\&x) \); // call with pointer (address of integer x)
   5. **The actual function code - Defining the function**
      - Declaration is same as the function prototype but now with braces and code
      - int foo( char* str ); //declared with a character pointer (2.A.3 above)
        { code block of function}
      - int foo( int x ); // call by value (copy of original) (2.A.2 above)
        { code block of function}

3. **Strings** in ’C’ language are arrays of characters
   A. Data type **char** is used for characters
   B. Strings are actually one-dimensional array of characters terminated by a null character '\0'.
   C. The compiler treats characters within quotes as a string
   D. Examples:
      1. char str[80]="Hello World"; // declaration: fixed dimension char array with first letters set
         printf (str); // printf is called with a pointer to str array parameter
      2. // compiler autodimension array in declaration
         char srg[]="This is an example\n"; // 18 chars + null character
         printf (srg); // printf is called with a pointer to srg array parameter
      3. // in this example string stored as an array with no name, just pointer to the string
         printf ("This is an example \n"); // printf is called with a pointer to unnamed char array

4. **Pointer Operators**:
   A. **Declare** the pointer using ‘*’ i.e. int* ptrToMyValue;
   B. Contents of: ‘&’: **dereference** the pointer - get the value at that address
   C. Address of: ‘&’: create a **reference** pointer - point to the variable address

5. **Example**: Function called with pointer parameter can change Parent original value:
   - **Declarations**:
      - int foo (int *x); // function prototype - Call by Reference
      - int x; // variable declaration
   - **Use the function (call it)**:
      - y = foo( &x ); // Call with int pointer
   - **Function code definition**:
      - int foo (int *x); // Call by reference
        { int tmp; // temporary value
          tmp = *x; //dereference x - get original contents of where x points;
          *x = 5; // dereference x again - change contents of where x points;
          return tmp; // return original value of x;
        }