1. **Arrays** contain objects and supply either a **Reference(pointer)** or **Dereference**(**Value**)
   A. Subscripts begin at 0: y = array[0];
   B. Array with *subscript* is a **Value**: y = array[3];
   C. Array *name only* is a **Reference** (pointer): py = array

2. **Functions:**
   A. A function has four parts: return data type, name, parameter data types and names, code
   B. Functions called by **value** present a *copy* to the function.
   C. Functions called by **reference,** a **pointer** (address) sent to the function.
   D. **Declare the function** – Function prototype
      1. **Return** data type is specified for function name
      2. Call by **Value**: variable
```
int foo( int x );  // called with a copy of an integer value
```
      3. Call by **Reference (pointer)**: - pointer or array name
```
int foo( int* apples);  // called with a pointer to an integer
int foo( int apples[] ); // called with a pointer to array first element
int foo( int apples[25] ); // called with a pointer to array first element
.............
int foo( char* str );  //called with a character pointer
int foo( char str[] );  //called with a character pointer
int foo( char str[80] ); //called with a character pointer
```
   E. **Call the function** – Within the program code
      1. By **Value**: - variable
         1. y = foo (x); // call child with *copy* of value of integer x
      2. By **Reference (pointer)**: - array
         1. status = foo(str); // call with *pointer* to char array (string) (see A.3 above)
         2. status = foo(apples); // call with *pointer* to integer array apples (see A.3 above)
   F. **Define the function** – the actual function code
      1. ***Same as the function prototype*** but with braces and code
```
int foo( char* str ) // call by reference (pointer)
{
    code block of function
}
```

3. **Strings** in 'C' language are arrays of characters
   A. Data type **char** is used for characters
   B. A string is an array of characters with *newline* (binary zero) as the last character.
   C. The compiler treats characters within quotes as a string
   D. Examples:
```
1   char str[80]="Hello World";  // declare a fixed dimension character array (string)
      printf (str); // called by Reference - pointer to a string
2  // compiler autodimension array with initialization declaration
    char srg[]="This is an example\n";  // 18 chars + newline
      printf (srg);  // called by Reference - pointer to a string
3  // in printf, string stored in memory by compiler with printf called by Reference
      printf ("This is an example \n"); // called by Reference - pointer to a string
```

4. **Pointer Operators:**
   A. Declare a pointer using '*': int* ptrToMyValue;
   B. Dereference the pointer – get the value at that address: int A = *ptrToMyValue
   C. Address of: '&': get a **reference** pointer - point to the value address: ptrToMyValue = &A;

5. **Change a Parent variable value:** Call by Reference (*pointer)* to modify a single variable
   A. **Declaration:**
```
1.  int foo (int *x); // Call by Reference
2.  int x;
```
   B. **Call** in parent code:
```
3.  y = foo( &x ); // Call with pointer
```
   C. **Function definition:**
```
4.  int foo (int *x); // Call by reference
5.  {
6.      int tmp;
7.      tmp = *x  //get contents of x;
8.      *x += 5;  // change contents of x;
9.      return tmp;  // return original value of x;
10. }
```