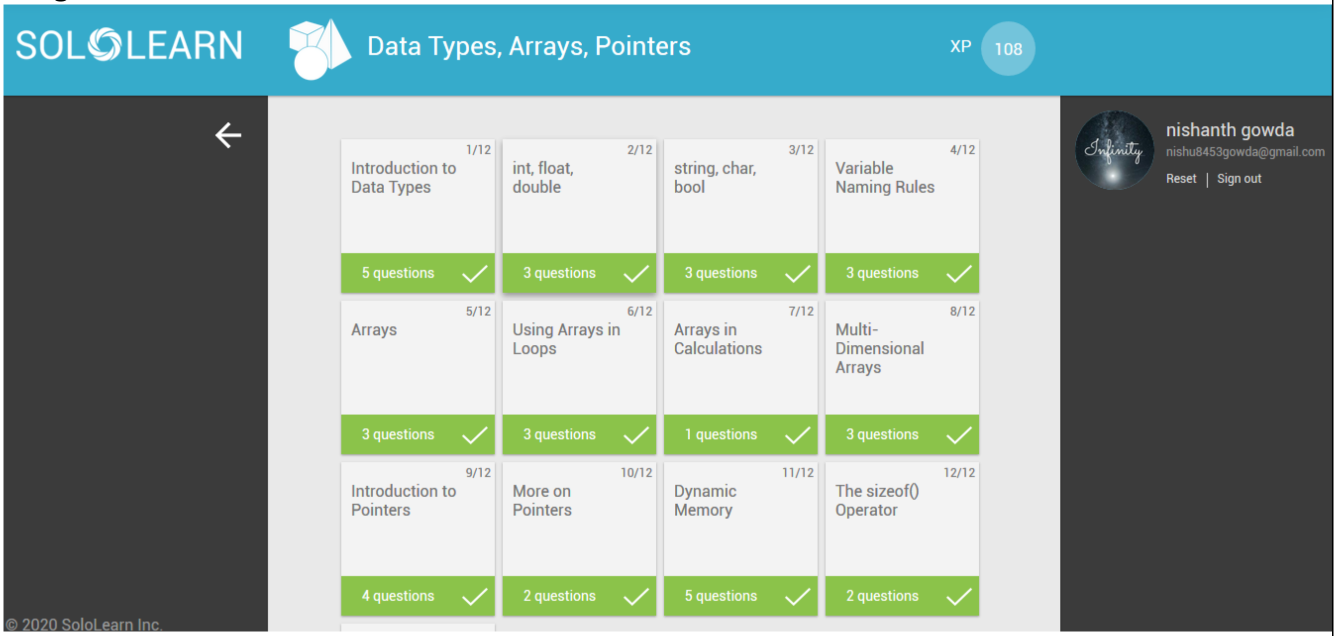
DAILY ASSESSMENT FORMAT

Date:	**23/June/2020**	**Name:**	**nishanth**
Course:	**C++ programming**	**USN:**	**4al17ec063**
Topic:	**1.data type** **2.array** **3.pointer** **4.dynamic memory**	**Semester & Section:**	**6th b**
GitHub Repository:	**nishanthvr**		

FORENOON SESSION DETAILS

Image of session



An **array** is used to store a collection of data, but it may be useful to think of an array as a collection of variables that are all of the **same type**.

Instead of declaring multiple variables and storing individual values, you can declare a single array to store all the values.

When declaring an array, specify its element types, as well as the number of elements it will hold.

For example: int a[5];

In the example above, variable **a** was declared as an array of five integer values [specified in square brackets].

You can initialize the array by specifying the values it holds: int b[5] = {11, 45, 62, 70, 88};

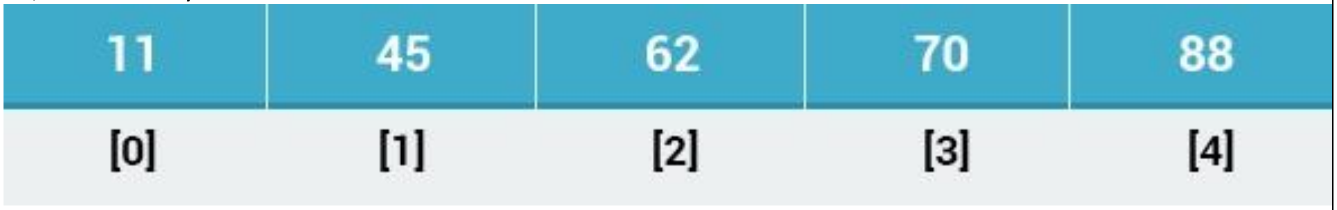
The values are provided in a **comma** separated list, enclosed in **{curly braces}**.

The number of values between braces { } must not exceed the number of the elements declared within the square brackets [ ].

Initializing Arrays

If you omit the size of the array, an array just big enough to hold the initialization is created.
For example: int b[] = {11, 45, 62, 70, 88};
This creates an identical array to the one created in the previous example.

Each element, or member, of the array has an **index**, which pinpoints the element's specific position.
The array's first member has the index of 0, the second has the index of 1.
So, for the array **b** that we declared above:



11	45	62	70	88
[0]	[1]	[2]	[3]	[4]

To access array elements, index the array name by placing the element's index in square brackets following the array name.
For example:

```
int b[] = {11, 45, 62, 70, 88};

cout << b[0] << endl;
// Outputs 11

cout<< b[3] << endl;
// Outputs 70
```

Pointers

Every variable is a **memory** location, which has its **address** defined.
That address can be accessed using the **ampersand (&)** operator (also called the address-of operator), which denotes an **address in memory**.

For example:

```
int score = 5;
cout << &score << endl;

//Outputs "0x29fee8"
```

A **pointer** is a variable, and like any other variable, it must be declared before you can work with it.
The **asterisk** sign is used to declare a pointer (the same asterisk that you use for multiplication), however, in this statement the asterisk is being used to designate a variable as a pointer.
Following are valid pointer declarations: int *ip; // pointer to an integer

```
double *dp; // pointer to a double
float *fp; // pointer to a float
char *ch; // pointer to a character
```

Just like with variables, we give the pointers a name and define the type, to which the pointer points to.
The asterisk sign can be placed next to the data type, or the variable name, or in the middle.

Dynamic Memory

For local variables on the **stack**, managing memory is carried out automatically.
On the **heap**, it's necessary to manually handle the dynamically allocated memory, and use the **delete** operator to free up the memory when it's no longer needed. **delete** pointer;
This statement releases the memory pointed to by **pointer**.

For example:

```
int *p = new int; // request memory
*p = 5; // store value

cout << *p << endl; // use value

delete p; // free up the memory
```