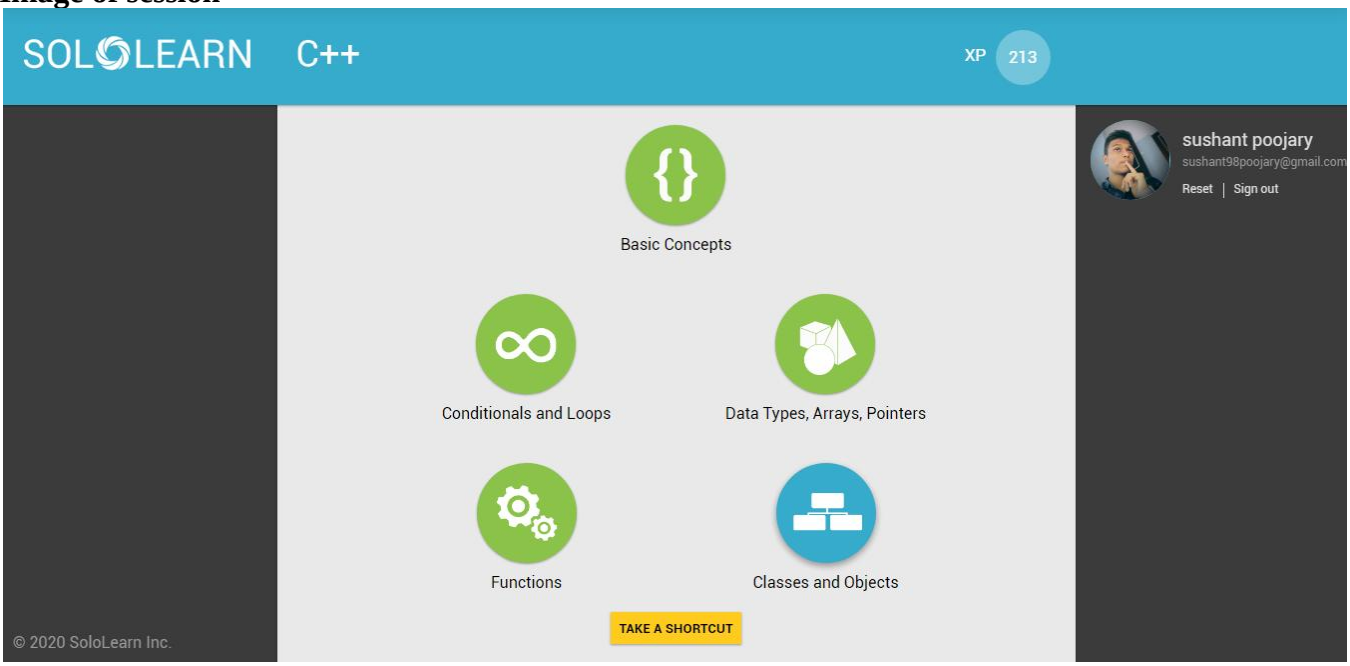
DAILY ASSESSMENT FORMAT

Date:	**23rd June 2020**	**Name:**	**Poojary Sushant**
Course:	**Solo Learning**	**USN:**	**4AL18EC400**
Topic:	**Data types, Arrays,Pointers,Functions**	**Semester & Section:**	**6th sem 'B"**
Github Repository:	**Sushant7026**		

FORENOON SESSION DETAILS

Image of session



Report –

Data Types

The operating system allocates memory and selects what will be stored in the reserved memory based on the variable's data type. The data type defines the proper use of an identifier, what kind of data can be stored, and which types of operations can be performed.

Numeric Data Types

Numeric data types include:
Integers (whole numbers), such as -7, 42.
Floating point numbers, such as 3.14, -42.67.

Strings & Characters

A string is composed of numbers, characters, or symbols. String literals are placed in double quotation marks; some examples are "Hello", "My name is David", and similar.
Characters are single letters or symbols, and must be enclosed between single quotes, like 'a', 'b', etc.

Booleans

The Boolean data type returns just two possible values: true (1) and false (0).

Arrays

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}.

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"
```

Pointers

A pointer is a variable, with the address of another variable as its value.
In C++, pointers help make certain tasks easier to perform. Other tasks, such as dynamic memory allocation, cannot be performed without using pointers.
All pointers share the same data type - a long hexadecimal number that represents a memory address.

Using Pointers

Here, we assign the address of a variable to the pointer.

```
int score = 5;
int *scorePtr;
scorePtr = &score;
cout << scorePtr << endl;
//Outputs "0x29fee8"
```

Functions

A function is a group of statements that perform a particular task.
You may define your own functions in C++.
Using functions can have many advantages, including the following:
- You can reuse the code within a function.
- You can easily test individual functions.
- If it's necessary to make any code modifications, you can make modifications within a single function, without altering the program structure.
- You can use the same function for different inputs.

The Return Type

The main function takes the following general form:

```
int main()
{
// some code
return 0;
}
```

A function's return type is declared before its name. In the example above, the return type is int, which indicates that the function returns an integer value.
Occasionally, a function will perform the desired operations without returning a value. Such functions are defined with the keyword void.

Defining a Function

Define a C++ function using the following syntax:

```
return_type function_name( parameter list )
{
body of the function
}
```

return-type: Data type of the value returned by the function.

function name: Name of the function.

parameters: When a function is invoked, you pass a value to the parameter. This value is referred to as actual parameter or argument. The parameter list refers to the type, order, and number of the parameters of a function.

body of the function: A collection of statements defining what the function does.

Defining a Function

As an example, let's define a function that does not return a value, and just prints a line of text to the screen.

```
void printSomething()
{
cout << "Hi there!";
}
```

Our function, entitled printSomething, returns void, and has no parameters.

Now, we can use our function in main().

```
int main()
{
printSomething();
return 0;
}
```

Function Arguments

There are two ways to pass arguments to a function as the function is being called.

By value: This method copies the argument's actual value into the function's formal parameter. Here, we can make changes to the parameter within the function without having any effect on the argument.

By reference: This method copies the argument's reference into the formal parameter. Within the function, the reference is used to access the actual argument used in the call. This means that any change made to the parameter affects the argument.

Passing by Value

By default, arguments in C++ are passed by value.

When passed by value, a copy of the argument is passed to the function.

Example:

```
void myFunc(int x) {
x = 100;
}
int main() {
int var = 20;
myFunc(var);
cout << var;
}
```

// Outputs 20

Passing by value: This method copies the actual value of an argument into the formal parameter of the function. In this case, changes made to the parameter inside the function have no effect on the argument.

Passing by reference: This method copies the reference of an argument into the formal parameter. Inside the function, the reference is used to access the actual argument used in the call. So, changes made to the parameter also affect the argument.