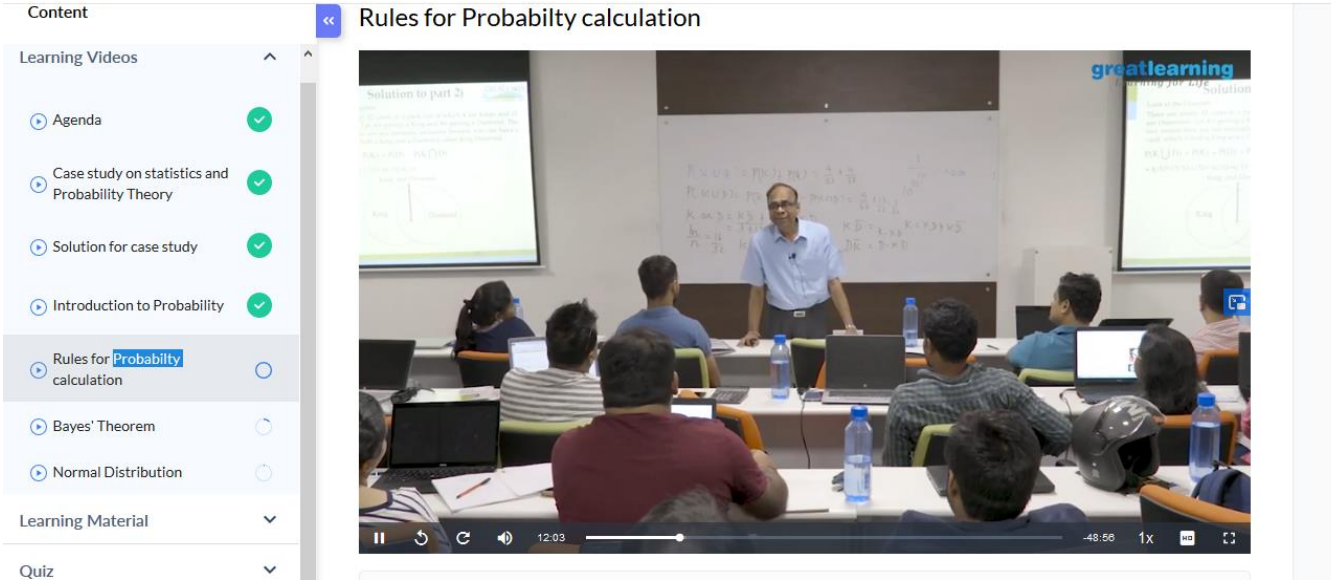
DAILY ASSESSMENT FORMAT

Date:	**16/June/2020**	**Name:**	**nishanth**
Course:	**Statistical Learning**	**USN:**	**4al17ec074**
Topic:	**1.introduction probability** **2.rules of probability calculation**	**Semester & Section:**	**6th b**
GitHub Repository:	**nishanthvr**		

FORENOON SESSION DETAILS
Image of session



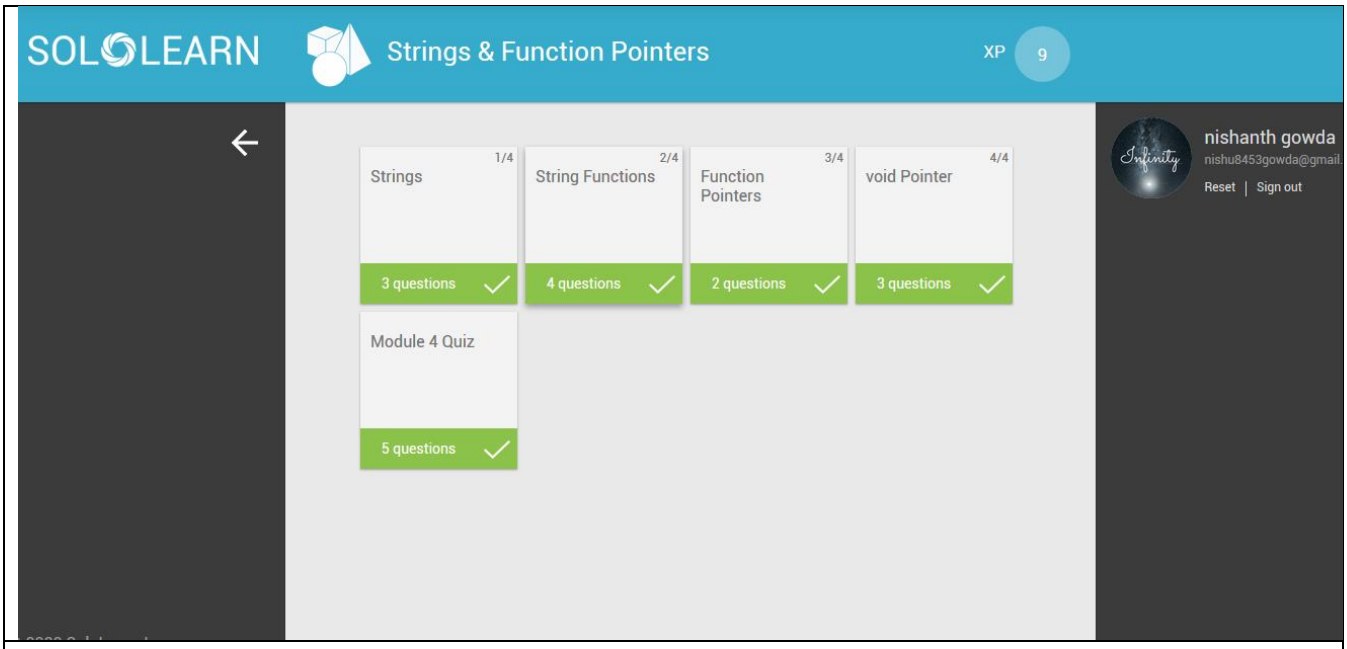
Basic Probability: Some Concepts

The probability that an event will occur is a number between 0 and 1. In other words, it is a fraction. It is also sometimes written as a percentage, because a percentage is simply a fraction with a denominator of 100. For more about these concepts, see our pages on Fractions and Percentages.

An event that is certain to occur has a probability of 1, or 100%, and one that will definitely not occur has a probability of zero. It is also said to be impossible.

- Two events are **mutually exclusive** or **disjoint** if they cannot occur at the same time.
- The probability that Event A occurs, given that Event B has occurred, is called a **conditional probability**. The conditional probability of Event A, given Event B, is denoted by the symbol P(A|B).
- The **complement** of an event is the event not occurring. The probability that Event A will not occur is denoted by P(A').
- The probability that Events A and B *both* occur is the probability of the **intersection** of A and B. The probability of the intersection of Events A and B is denoted by P(A ∩ B). If Events A and B are mutually exclusive, P(A ∩ B) = 0.
- The probability that Events A or B occur is the probability of the **union** of A and B. The probability of the union of Events A and B is denoted by P(A ∪ B) .
- If the occurrence of Event A changes the probability of Event B, then Events A and B are **dependent**. On the other hand, if the occurrence of Event A does not change the probability of Event B, then Events A and B are **independent**.

Date:	**17/June/2020**	**Name:**	**nishanth**
Course:	**C programming**	**USN:**	**4al17ec074**
Topic:	**1.Structures and union 2.memory management**	**Semester&Section :**	**6th b**
Git hub repository	**nishanthvr**		
AFTERNOON SESSION DETAILS			



Structures

A **structure** is a **user-defined data type** that groups related variables of different data types.

A structure **declaration** includes the keyword **struct**, a **structure tag** for referencing the structure, and curly braces { } with a list of variable declarations called **members**.
For example:struct course {
int id;
char title[40];
float hours;
};
This struct statement defines a new data type named **course** that has three members.
Structure members can be of any data type, including basic types, strings, arrays, pointers, and even other structures, as you will learn in a later lesson.
Do not forget to put a semicolon after structure declaration.
A structure is also called a **composite** or **aggregate** data type. Some languages refer to structures as **records**.

Structures With Unions

Unions are often used within structures because a structure can have a member to keep track of which union member stores a value.
For example, in the following program, a vehicle struct uses either a vehicle identification number (VIN) or an assigned id, but not both:
typedef struct {
char make[20];
int model_year;
int id_type; /* 0 for id_num, 1 for VIN */
union {
int id_num;
char VIN[20];
} id;
} vehicle;

```
vehicle car1;
strcpy(car1.make, "Ford");
car1.model_year = 2017;
car1.id_type = 0;
car1.id.id_num = 123098;
```

Memory Management

Understanding memory is an important aspect of C programming. When you declare a variable using a basic data type, C automatically allocates space for the variable in an area of memory called the **stack**.

An int variable, for example, is typically allocated 4 bytes when declared. We know this by using the **sizeof** operator:

```
int x;
printf("%d", sizeof(x)); /* output: 4 */
```

As another example, an array with a specified size is allocated **contiguous blocks** of memory with each block the size for one element:

```
int arr[10];
printf("%d", sizeof(arr)); /* output: 4
```

So long as your program explicitly declares a basic data type or an array size, memory is automatically managed. However, you have probably already been wishing to implement a program where the array size is undecided until runtime.

Dynamic memory allocation is the process of allocating and freeing memory as needed. Now you can prompt at runtime for the number of array elements and then create an array with that many elements. Dynamic memory is managed with pointers that point to newly allocated blocks of memory in an area called the **heap**. In addition to automatic memory management using the stack and dynamic memory allocation using the heap, there is **statically managed data** in **main memory** for variables that persist for the lifetime of the program.