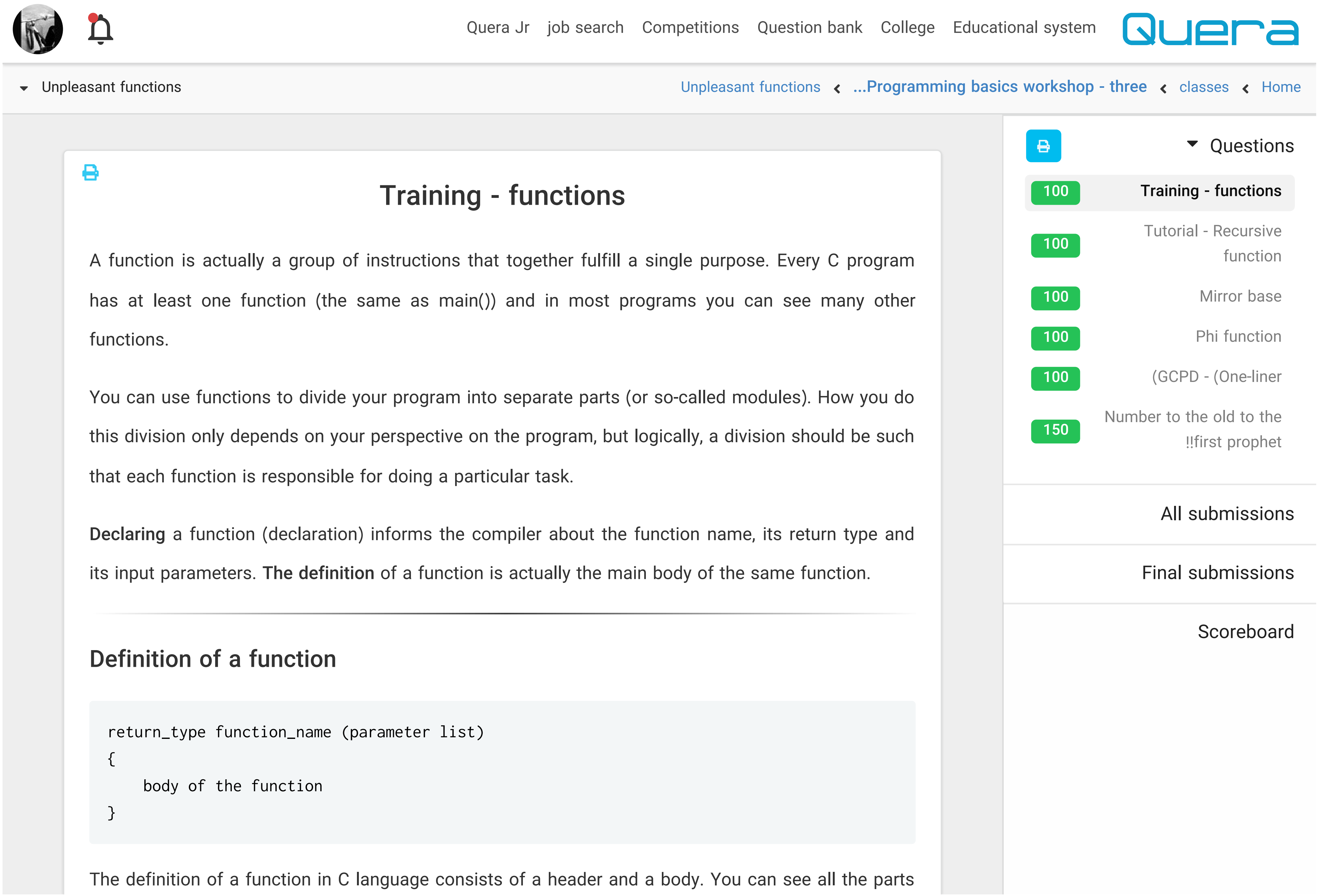

Training - functions

A function is actually a group of instructions that together fulfill a single purpose. Every C program has at least one function (the same as main()) and in most programs you can see many other functions.

You can use functions to divide your program into separate parts (or so-called modules). How you do this division only depends on your perspective on the program, but logically, a division should be such that each function is responsible for doing a particular task.

Declaring a function (declaration) informs the compiler about the function name, its return type and its input parameters. **The definition** of a function is actually the main body of the same function.

Definition of a function

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

The definition of a function in C language consists of a header and a body. You can see all the parts of a function explained below:

- Return Type:

A function "can" have a return value. return_type is actually the data type that the function will return. Some functions do what they are assigned without returning a return value. In this case, return_type will be equal to keyword void.

- Function Name:

This part actually shows the name of the function. The name of the function and the list of input parameters of the function together form the signature of the function. (Function Signature)

- Parameters:

Each parameter is actually like a placeholder. When a function is called, we pass a value as a parameter to the function. This value is called the main parameter or an argument. The parameter list actually refers to the type, order and number of parameters of a function. Parameters are optional for a function, in fact we can have function without input parameters.

- Function Body:

The body of a function is actually a set of commands that define the main purpose of a function.

For example, the following function takes two arguments num1 and num2 and returns the maximum value of these two.

```
/* function returning the max between two numbers */
```

```
int max(int num1, int num2) {

   /* local variable declaration */
   int result;

   if (num1 > num2)
      result = num1;
   else
      result = num2;

   return result;
}
```

Declaration of functions

As mentioned, declaring a function informs the compiler about the name of the function and how to call the function. The main body of the function can be defined separately. The declaration of a function has the following parts:

```
return_type function_name (parameter list);
```

For the above example (max() function), the declaration of the function is as follows:

```
int max(int num1, int num2);
```

Parameter names are not necessary in the declaration of a function, and only their types are important. Therefore, the following statement is also a true statement:

```
int max(int, int);
```

Declaration (or signature) of a function is required when you have defined a function in a source file and want to use it in another file. In this case, you must write the function declaration at the top of the file in which you called the function.

Calling a function

When you create a function in C, you also provide a definition for it, which is actually the agenda of that function. To use the function to perform that task, you must call it. When a program calls a function, program control is passed to the called function. When the function is called, it executes the task defined for it, and when it reaches the end of the function definition or a return command, it returns control to the main program. To call a function, it is enough to pass the necessary parameters (in the order specified in the function declaration) to the function and use the return value (if any) using the function name and these parameters.

Pay attention to the following example:

```
#include <stdio.h>

/* function declaration */
int max(int num1, int num2);

int main () {

```

```c
   /* local variable definition */
   int a = 100;
   int b = 200;
   int ret;

   /* calling a function to get max value */
   ret = max(a, b);

   printf( "Max value is : %d\n", ret );

   return 0;
}

/* function returning the max between two numbers */
int max(int num1, int num2) {

   /* local variable declaration */
   int result;

   if (num1 > num2)
      result = num1;
   else
      result = num2;

   return result;
}
```

After compiling and running, this program will produce the following output:

```
Max value is : 200
```

Arguments of a function

If a function receives input arguments, it must declare variables that accept the values of those arguments. We call these variables **formal** function parameters. Formal parameters, like any other local variable inside a function, are created when the function is called and destroyed when it exits. Arguments are passed in two general ways, called Call by value and Call by reference. Call by reference will be discussed more later, but for now, all functions in C are called as call by value, in this way, instead of passing the reference of a variable, we pass its value to the function. (Therefore, due to the fact that formal parameters exist only during the operation of the function, you cannot, for example, use Call by value in a function (for example, swap) to exchange the value stored in two variables a and b why?)

Exercise : Try to write a swap function and use the Call by value method to exchange the values of the two variables a and b that are defined in main, and justify the reason for the failure of this action to yourself (by debugging the program).

POST AN ANSWER TO THIS QUESTION

.The training period is over