Title: Basics of Functions in Object-Oriented Programming using C++

Introduction:
Object-oriented programming (OOP) utilizes functions to encapsulate code and promote reusability. This theory provides a concise overview of function fundamentals in C++ within the context of OOP.

Function Basics:
Functions in C++ are self-contained blocks of code that execute specific tasks. They encapsulate statements, accept input parameters, and may return values, enabling code reusability and organization.

Function Declaration and Definition:
Functions are declared by specifying their return type, name, and input parameters. The function definition contains the actual code implementation. For example:

```
int add(int a, int b) {
   return a + b;
}
```

Function Invocation:
Functions are called or invoked by using their name followed by parentheses. Return values can be assigned or used directly in expressions. For example:

```
int result = add(5, 3);
```

Parameters and Arguments:
Functions use parameters to accept input values during invocation. Parameters allow for data passing and usage within the function's scope. Arguments are the values passed to parameters. For example:

```
void greet(const std::string& name) {
   std::cout << "Hello, " << name << "!" << std::endl;
}
greet("Alice");
```

Return Values:
Functions can return values using the return statement. The return type must match the declared return type of the function. For example:

```
int multiply(int a, int b) {
   return a * b;
}
int result = multiply(4, 5);
```

Function Overloading:
C++ supports function overloading, allowing multiple functions with the same name but different parameter lists. This improves flexibility and code readability. For example:

```
void print(int num) {
   std::cout << "Printing an integer: " << num << std::endl;
}
void print(const std::string& text) {
   std::cout << "Printing a string: " << text << std::endl;
}
print(42);
print("Hello, World!");
```

Conclusion:
Functions are vital in OOP using C++. They encapsulate code, promote reusability, and aid in organizing programs. Understanding function basics such as declaration, definition, invocation, parameters, return values, and function overloading allows for the creation of modular and maintainable code in OOP.