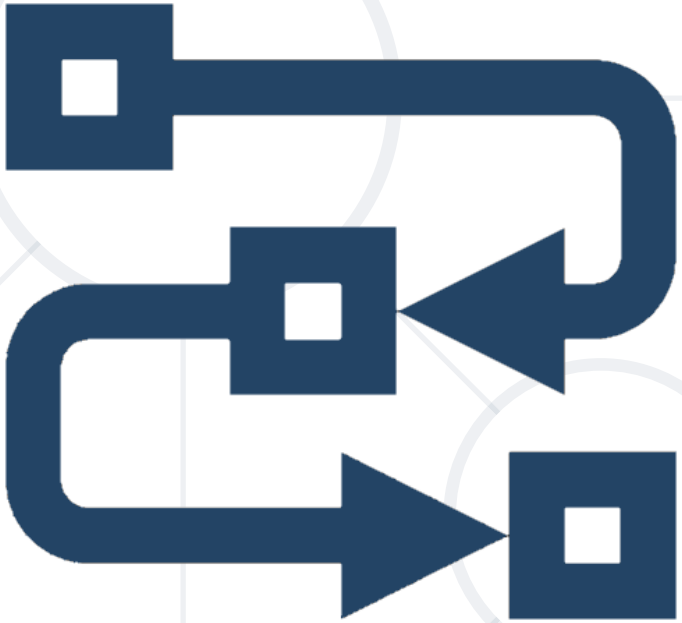
Functions

Defining and Using C++ Functions



SoftUni Team

Technical Trainers

Software University

SoftUni

Software University

https://softuni.bg

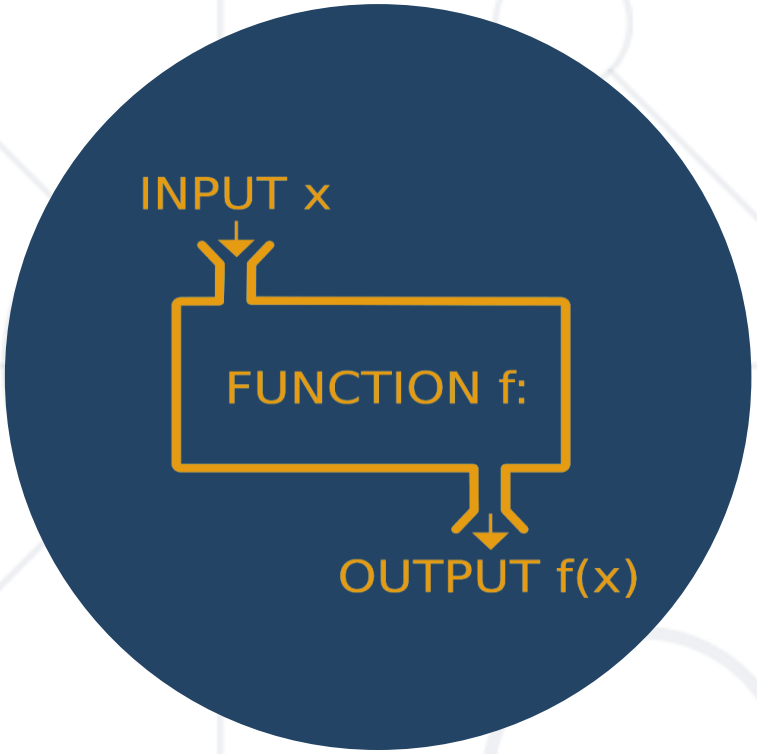
Table of Contents

Have a Question?

Software University

sli.do

#cpp-fundamentals



Functions

Calling, Defining, Implementing

What is a Function?

- **Named block of code**, that performs a specific task
- Can take parameters and return a value
- Sample function **definition**:

```
void printHelloWorld()
{
  cout << "Hello World!" << endl;
}
```

Function named `printHelloWorld`

Function **body** always surrounded by { }

- Also known as methods
- `main()` **is a function**

Why Use Functions?

- More **manageable programming**
  - Splits large problems into small pieces
  - Better organization of the program
  - Improves code readability
  - Improves code understandability
- Avoiding **repeating code**
  - Improves code maintainability
- Code **reusability**
  - Using existing methods several times

Declaring and Calling Functions

Declaring Functions

- Declaration – function's name, return type and parameters
  - Can be separate from definition (which includes the code block)
- Parameters: empty, single, or several separated by ,

Type

Function name

Parameters

```
void PrintNumber(int number)
{
   cout << number << endl;
}
```

Function body

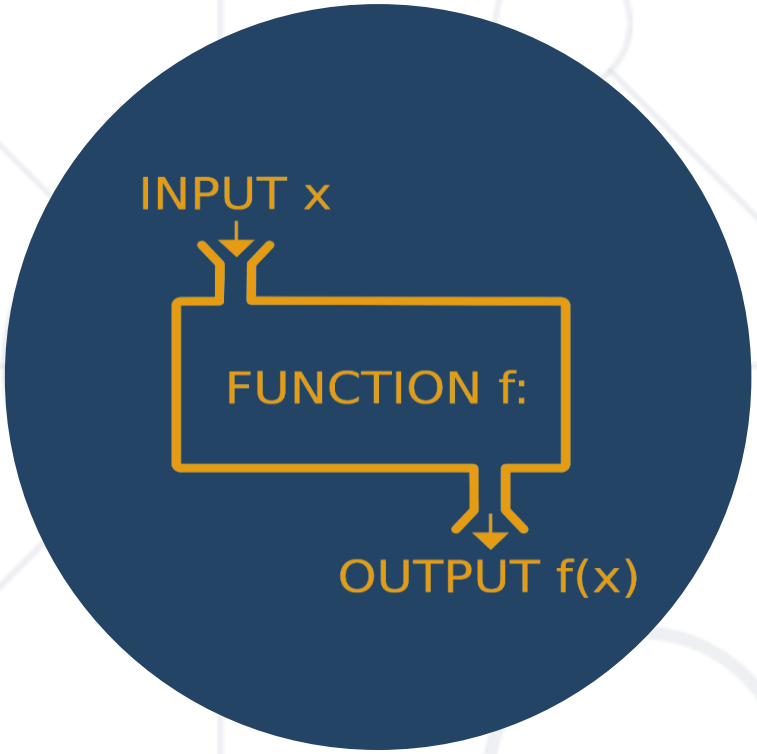
Calling Functions

- Using functions is almost like using variables, however:
  - You write **( )** after them, which could contain parameters
- Most functions return a value – you can use it in an expression
  - **void** functions don't have values

```
void HelloWorld()
{
    std::cout << "Hello World!" << std::endl;
}

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

Declaring and Calling Functions

LIVE DEMO

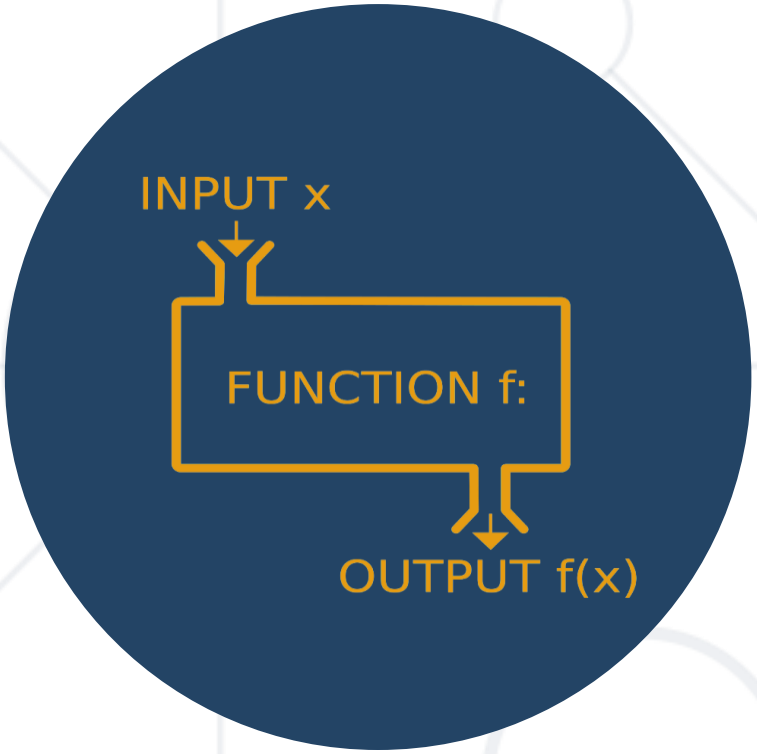


Declaring vs. Defining Functions

Declaring vs. Defining Functions

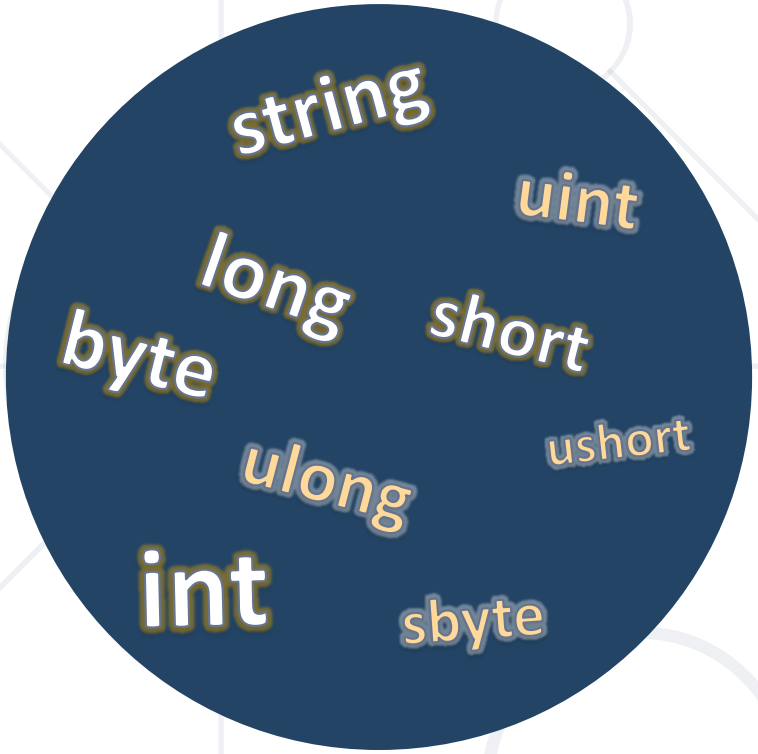
- Declaration – tells the compiler there is certain a function
  - Can be anywhere
  - Can appear multiple times
  - Same visibility rules as for variables
- Definition – function's execution
- Can be declared but not defined – compilation error if called

```
#include<iostream>
using namespace std;

void HelloWorld();

int main() {

HelloWorld();
    return 0;
}

void HelloWorld()
{
cout << "Hello World!" << endl;
}
```



Declaring vs. Defining Functions

LIVE DEMO



Functions with Parameters

Function Parameters

- Function **parameters** can be of **any data type**
- Parameters are just variables living in the function's block

```
void printNumbers(int start, int end)
{
  for (int i = start; i <= end; i++)
  {
    std::cout << i << std::endl;
  }
}
```

Multiple parameters separated by comma

- Call the method with certain values (**arguments**)

```
int main()
{
  printNumbers(5, 10);
  return 0;
}
```

Passing arguments when called

Parameters & Default Values

- Parameters with default values can be omitted by the caller
  - If omitted are initialized with the default value
  - Must be last in the parameter list

```
#include <iostream>
void CountNumbers(int a = 1, int b = 10)
{
    for( int i = a; i <= b; i++ )
    {
        std::cout << i << std::endl;
    }
}
int main()
{
    CountNumbers(5, 10);
    return 0;
}
```

Returning Values from Functions

Returning Values from Functions

- The **return** keyword immediately stops the function's execution
- Returns the specified value
  - Non-**void** functions must have a **return** followed by a value

```
int getMax(int a, int b) {
    if (a > b) {
        return a;
    }
    return b;
}
```

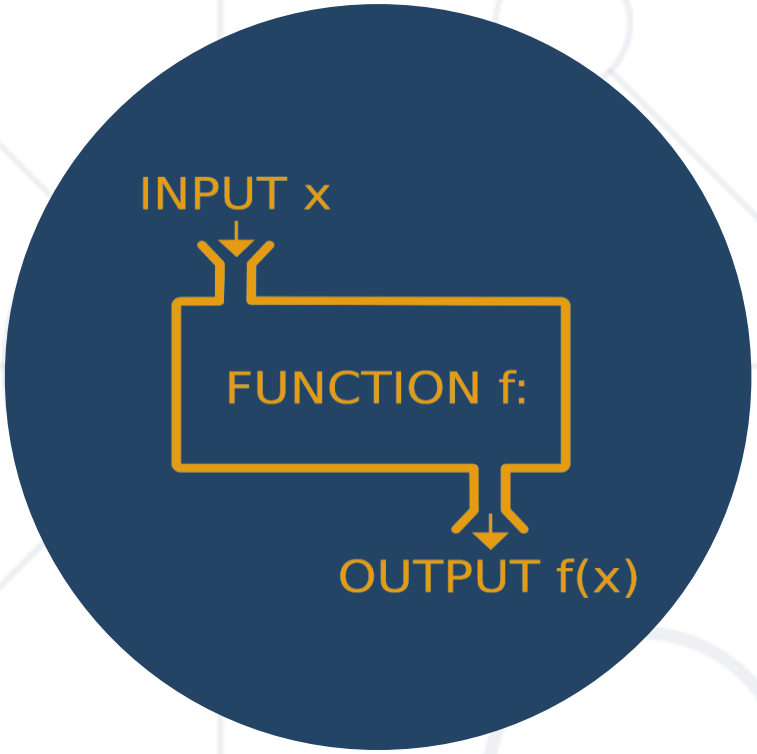
Using the Return Values

- Return value can be:
  - **Assigned** to a variable:

```
int max = getMax(5, 10);
```

  - **Used** in expression:

```
double total = getPrice() * quantity * 1.20;
```



Parameters and Returning Values

LIVE DEMO

Overloading Functions

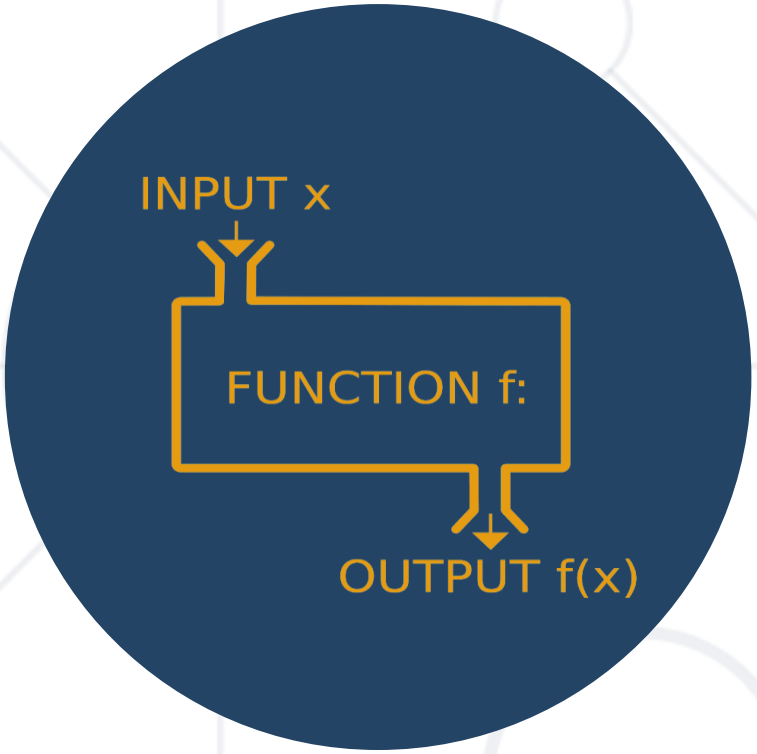
Overloaded Functions

- Using the same function **name** and **return type** but with different parameter list
  - Different number or types of parameters

```
int getMax(int a, int b) {
    if (a > b) {
        return a;
    }
    return b;
}

int getMax(int a, int b, int c) {
    return getMax(a, getMax(b, c));
}
```

Overloading Functions

LIVE DEMO

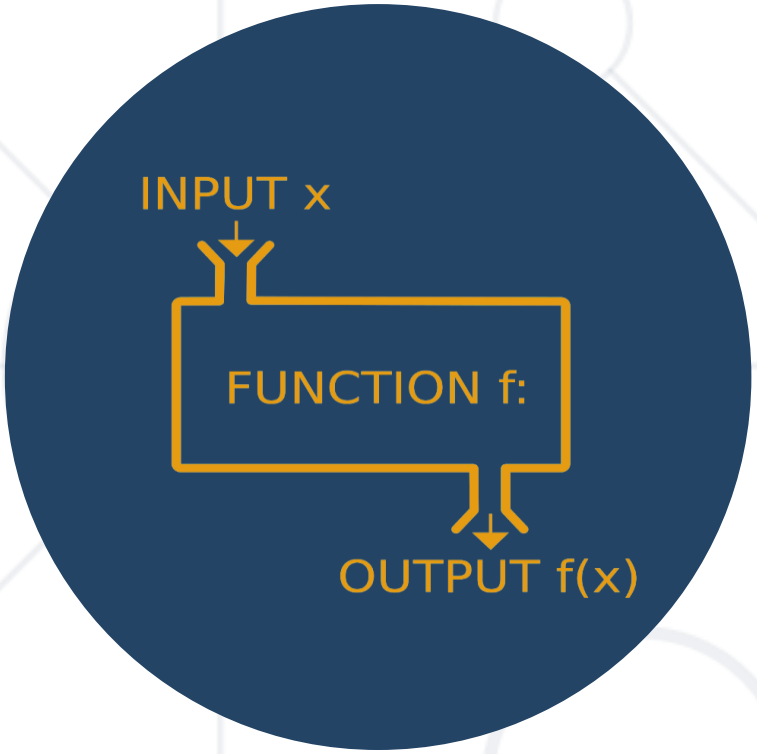


Static Variables Inside Functions

static Variables Inside Functions

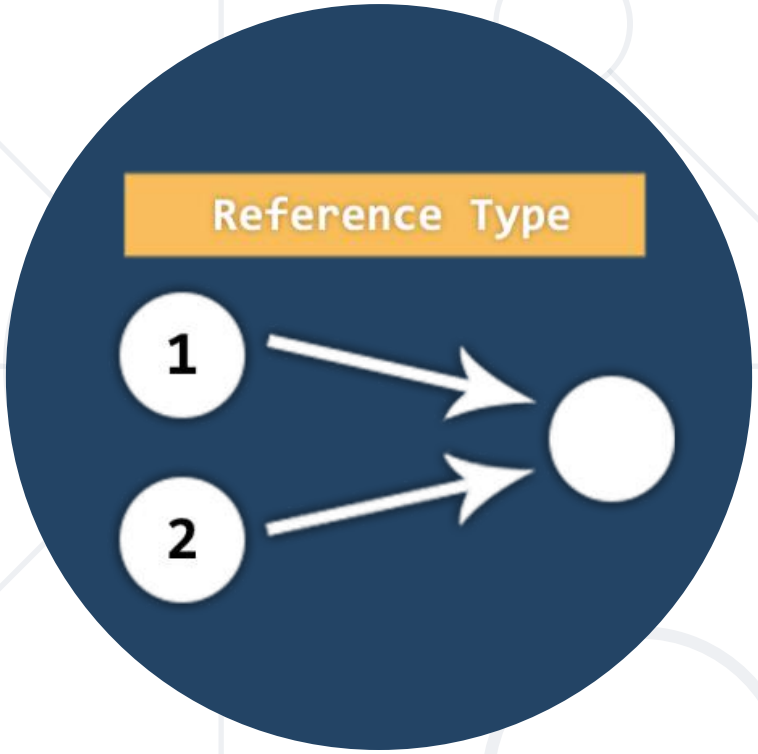
- **static** variables live through entire program, initialized once
- **static** variables can be used inside functions to track state
  - E.g. how many times a function was called

```
void CountNumbers( int a = 1, int b = 10 )
{
    static int num = 0;
    for( int i = a; i <= b; i++ )
    {
        cout << i << endl;
        num++;
    }
    cout << "Static int -> " << num << endl;
}
```



Static Variables Inside Functions

LIVE DEMO



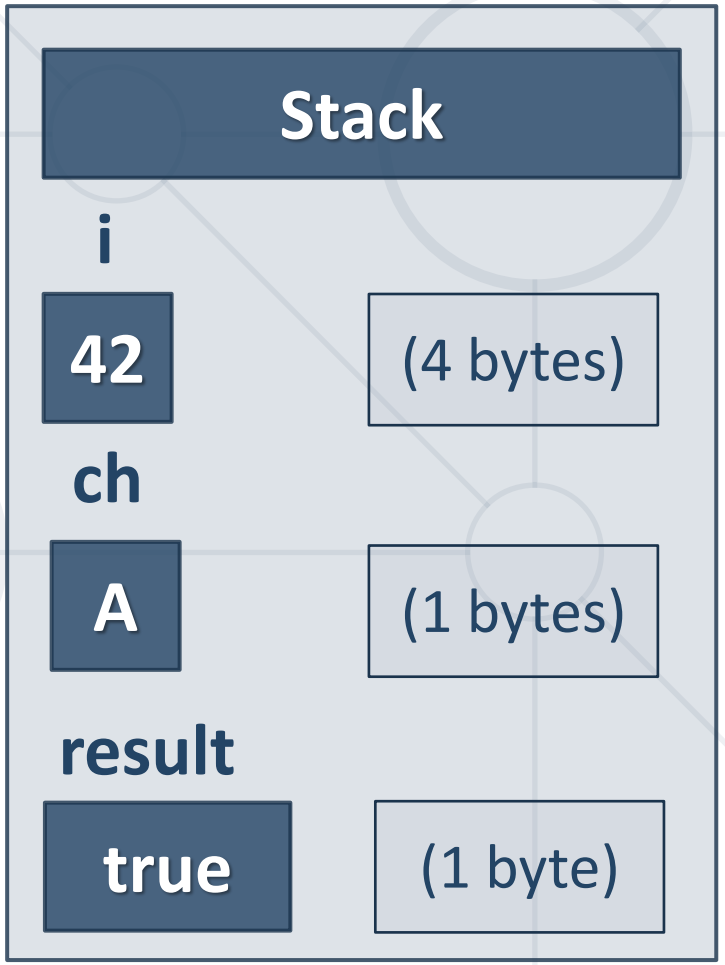
Value vs. Reference Types

Memory Stack and Heap

Value Types

- **Value type** variables hold directly their value
  - **int, float, double, bool, char…**
- Each variable has its own **copy** of the **value**

```
int i = 42;
char ch = 'A';
bool result = true;
```



Reference Types

- **Reference type** variables hold a reference (pointer / memory address) of the value itself
- Two reference type variables can **reference** the **same variable**
  - Operations on both variables access/modify **the same data**

Value vs. Reference Types

pass by reference

cup =

fillCup( )

pass by value

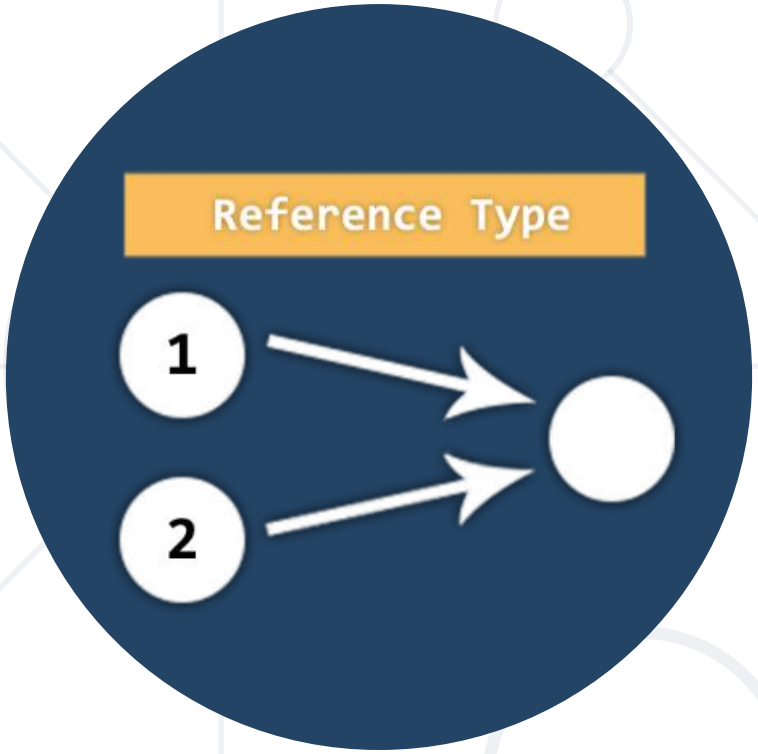
cup =

fillCup( )

Passing By Value vs. Passing By Reference

- Parameters are normally copies of their originals
  - Changing them does NOT change the caller's variables
  - "Passing by value"
- To access the caller's variables directly, use references
  - Syntax: `DataType& param`
  - "Passing by reference"

```
int square(int num) {
    num = num * num;
    return num;
}
void swap(int& a, int& b) {
    int oldA = a; a = b; b = oldA;
}

int main() {
    int x = 5;
    std::cout << square(x); //25
    std::cout << x; //5
    int y = 42;
    swap(x, y);
    std::cout << x; //42
    return 0;
```



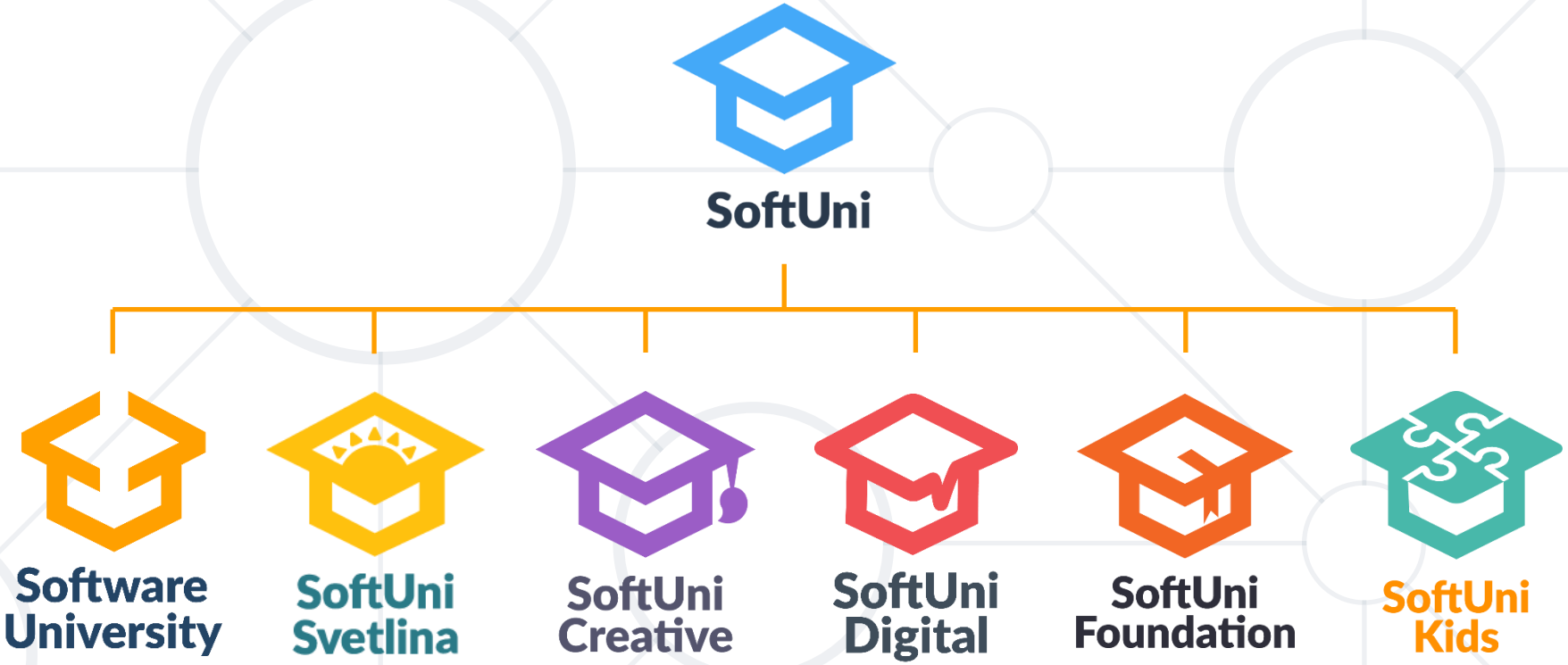
Value vs. Reference Types

LIVE DEMO

Summary

- Break large programs into simple **functions** that solve small sub-problems
- Functions consist of **declaration** and **body**
- Functions are called by their **name** + **()**
- Functions can accept **parameters**
- Functions can **`return`** a value or nothing (**`void`**)

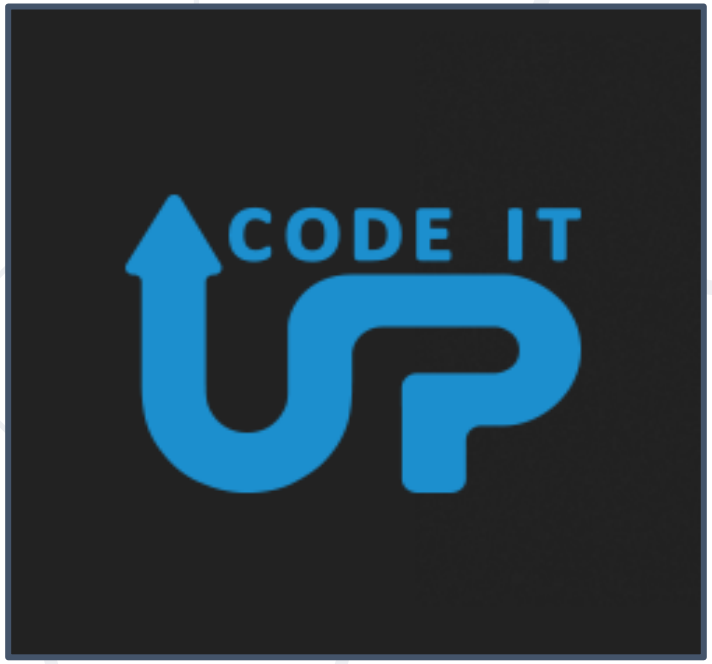
Questions?



SoftUni Diamond Partners

Software University

Educational Partners



VIRTUAL RACING SCHOOL

License

- This course (slides, examples, demos, exercises, homework, documents, videos and other assets) is **copyrighted content**
- Unauthorized copy, reproduction or use is illegal
- © SoftUni – https://about.softuni.bg/
- © Software University – https://softuni.bg

Trainings @ Software University (SoftUni)

- Software University – High-Quality Education, Profession and Job for Software Developers
  - softuni.bg, about.softuni.bg
- Software University Foundation
  - softuni.foundation
- Software University @ Facebook
  - facebook.com/SoftwareUniversity
- Software University Forums
  - forum.softuni.bg