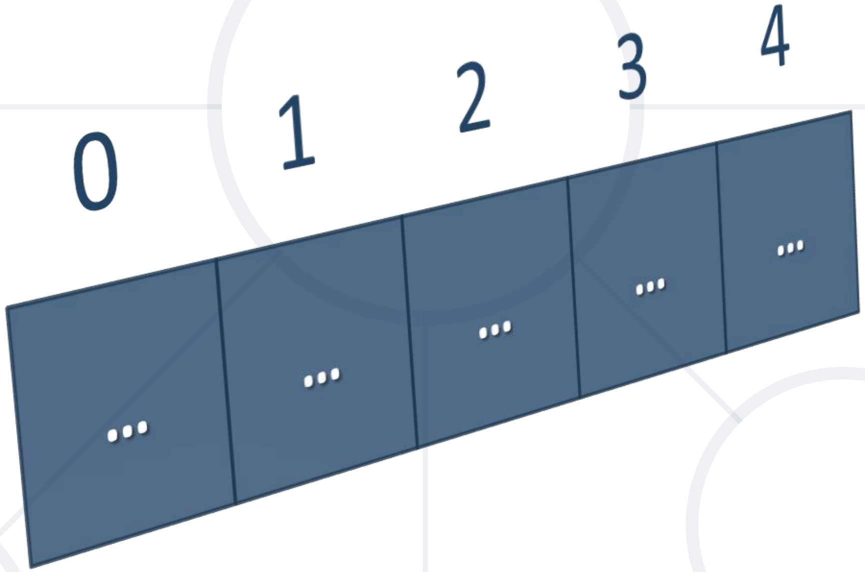
Vectors, Lists and Iterators



SoftUni Team

Technical Trainers

Software University

Software University

https://softuni.bg

SoftUni

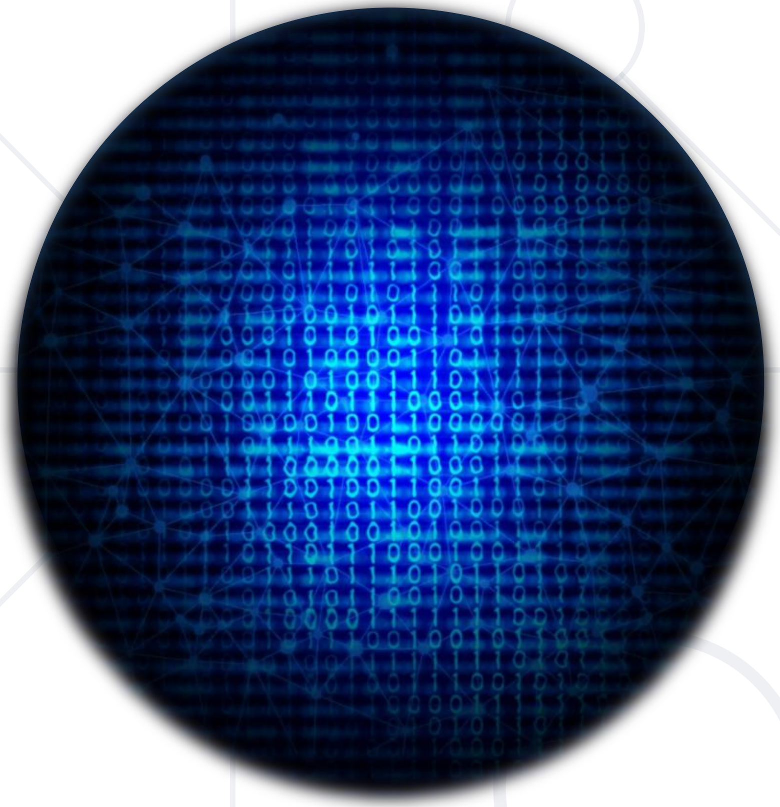
Table of Contents

1. Data Structures and Complexity
2. Vectors
3. Iterators
4. Lists

Have a Question?

sli.do

#cpp-fundamentals



Data Structures and Complexity

Classifying Data Containers by Operation

Data Structures

- Data Structures organize data for efficient access
  - Different data structures are efficient for different use-cases
  - Essentially: a data container + algorithms for access
- Some of the common data structures in Computer Science:
  - **Arrays** – fast access by index, **constant** / **dynamic** size
  - **Linked-list** – fast **add** / **remove** at any position, no index access
  - **Map** / **Dictionary** – contains **key** / **value** pairs, fast access by key

Complexity 101

- Complexity in Computer Science describes performance
  - How fast an algorithm runs and How much memory it consumes
  - Based on the size of the input data – usually denoted as **N**
  - We usually care about the worst-case performance
- How do we measure complexity?
  - **Time** = number of basic steps, **Memory** = number of elements
- Complexity is usually denoted by the **Big-O notation**
  - How much the number of steps grows compared to input size

Complexity 101

- We usually care about **X** orders of magnitude, not **+X** or ***X**
  - `O(N+3) == O(2N) == O(N)`, i. e. we care about the **N** parts

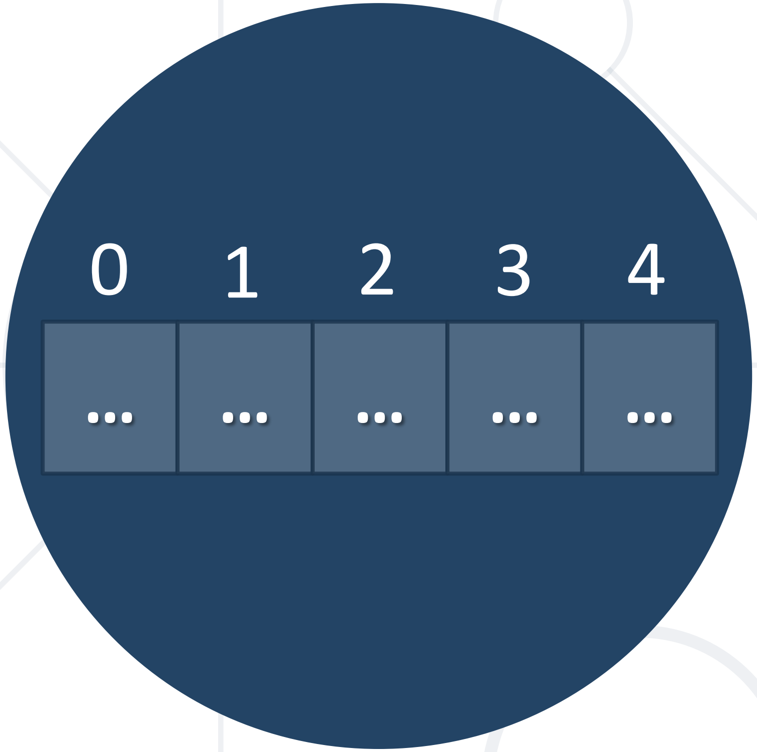
If something takes 1 million or 2 million years, it's the "million" that bothers you, not the "1" or the "2"

- **O(1)** – "constant" time / memory – input size has no effect
- **O(log(N))** – logarithmic – complexity grows as log(input) grows
- **O(N)** – linear – complexity grows as input grows
- **O(N^2)**, **O(N^3)**, … – quadratic, cubic, … – complexity grows with square/cube/etc. of input size
- **O(2^N)**, **O(3^N)**, … – exponential – this is a monster

Data Structure Performance 101

- If N is the number of elements in the container (the `.size()`):

	vector	list	map, set	unordered_map, unordered_set
access ith	O(1)	O(i)	O(i)	---
find(V)	O(N)	O(N)	O(log(N))	O(1) (usually)
insert(V)	O(1) at end (usually), O(N) otherwise	O(1)	O(log(N))	O(1) (usually)
erase(V)	O(1) at end (usually), O(N) otherwise	O(1)	O(log(N))	O(1) (usually)
Getting a sorted seq uence	O(N*log(N)) (using std::sort algorithm)	O(N + N*log(N)) (using .sort() method)	O(N) (by just iterating)	---



Vectors

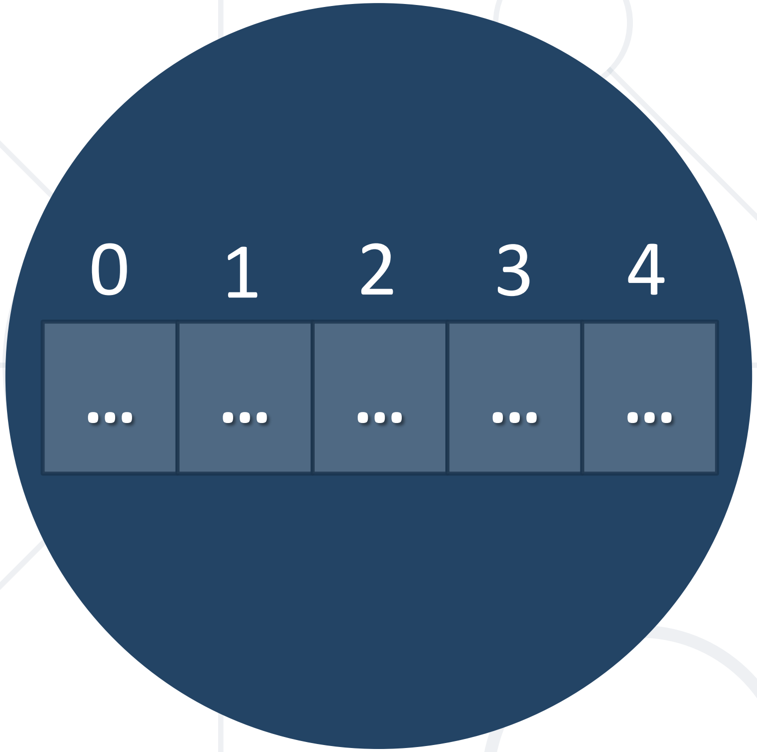
C++ Dynamically-Sized Arrays

STL Vector Basics

- The C++ `std::vector` class is a resizable array
  - Has normal array-like access – `[]` operator
  - Size is known (`.size()`)
  - Can add elements (`.push_back()`)
- `#include<vector>`
- Acts like a normal variable
  - Can be assigned like a normal variable
  - Can be returned from a function

`std::vector`

- Has all array operations
- Changes size automatically when elements added
- `push_back()` complexity is *amortized* `O(1)`
  - Usually takes `O(1)` time, occasionally takes `O(N)` time
  - Slow ~10 times out of ~1000, ~32 times out of ~4 billion, etc.
- Fast access `O(1)` to any element (random index access)
  - `arr[0] = 69; arr[15] = 42;`



Initializing STL Vectors

Initializing a Vector

- Declaration Syntax:

```
std::vector<T> name;
```

- The vector is initially empty – items need to be added
  - Call **push_back(T element)** on the vector to add elements

```
std::vector<int> myVector;
for (int i = 0; i < 100; i++)
{
    myVector.push_back( i + 10 );
}
```

- Can be initialized directly in C++11 with **{}** syntax
  - `std::vector<int> numbers {13, 42, 69};`
  - `std::vector<int> numbers = {13, 42, 69};`

Initializing STL Vectors

LIVE DEMO

Returning STL Vectors from Functions

Returning STL Vectors from Functions

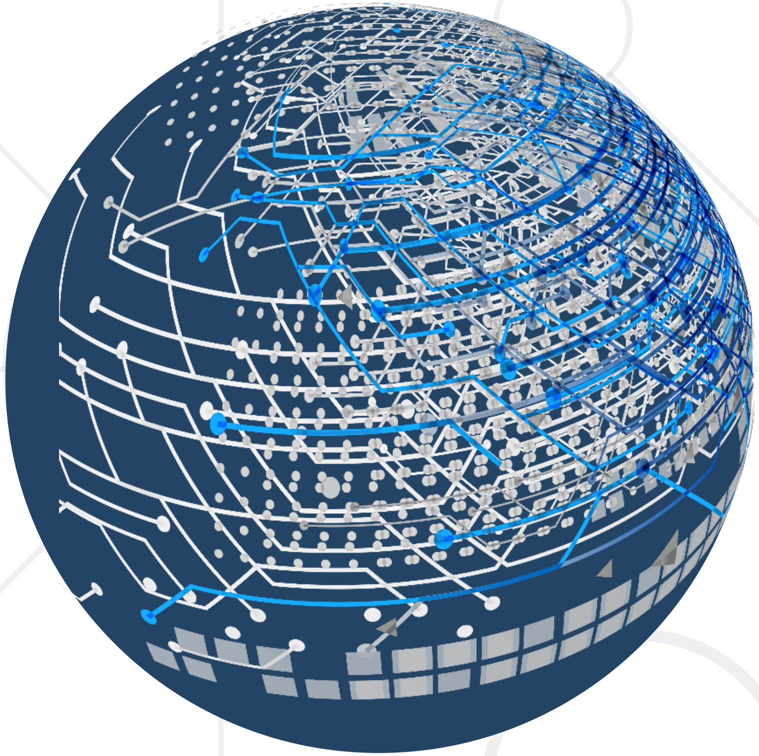
```
void print(vector<double> numbers) {
    for (int number : numbers) {
        cout << number << " "
    }
    cout << endl;
}
vector<double> getSquareRoots(int from, int to) {
    vector<double> roots;
    for (int i = from; i <= to; i++) {
        roots.push_back(sqrt(i));
    }
    return roots;
}
int main() {
    print(getSquareRoots(4, 25));
    return 0;
}
```

Vectors acts as normal variables when returned

Function returns a copy

Returning STL Vectors from Functions

LIVE DEMO



size_t and size_type

size_t and size_type

- Alias of one of the integer types
  - unsigned long int or unsigned long long int
  - Able to represent the size of any object in bytes
  - sizeof() returns size_t
- Each STL container offers a similar ::size_type
  - A good practice is to use it instead of int for sizes, positions, etc.

```
for (vector<int>::size_type i = 0; i < nums.size(); i++) {
  cout << nums[i] << endl;
}
```

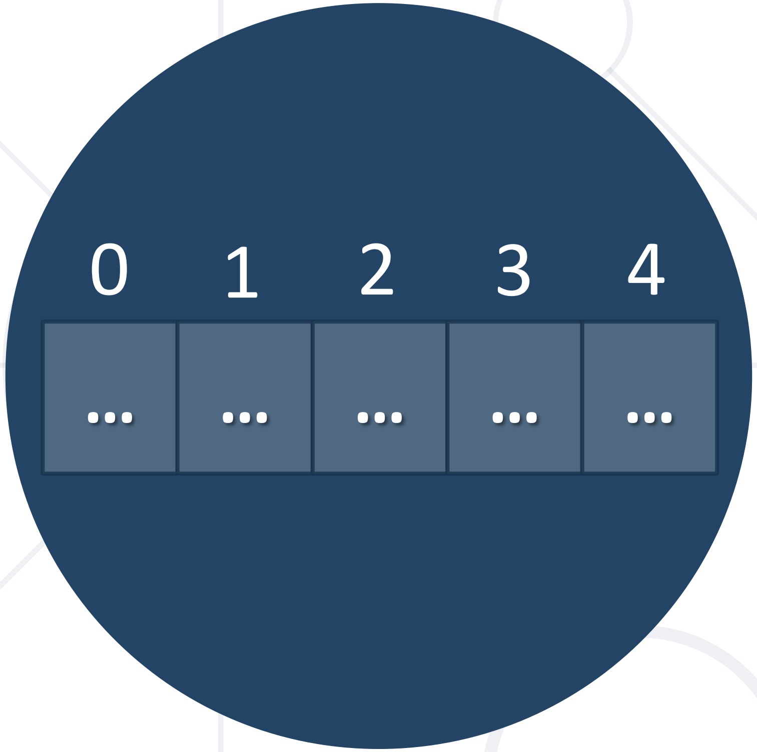
size_t and size_type

LIVE DEMO

Iterators

Container Iterators

- STL Iterators are things that know how to traverse a container
  - `operator++` - moves iterator to the next element
  - `operator*` - accesses the element
  - `operator->` - same as `operator.` on the element
- Each container has an iterator (`std::vector<T>::iterator`)
- Each container has `begin()` and `end()` iterators
  - `begin()` points to first element, `end()` to **after** last
  - Range-based `for`-loop uses them to work on **any** container



Using Iterators with Vectors

Using Iterators with Vectors

- Using iterators on **vectors** is almost the same as using indexes
- To go through a vector:
  - Start from **begin()**, move with **++** until you reach **end()**
  - Access the current element with *****

```
vector<int> nums {42, 13, 69};
for (vector<int>::iterator i = nums.begin(); i != nums.end(); i++) {
  cout << *i << endl;
}
// Equivalent code
for (vector<int>::size_type i = 0; i < nums.size(); i++) {
  cout << nums[i] << endl;
}
```

Using Iterators (1)

- Example: Change each element in the vector by dividing it by 2

```
vector<int> numbers {42, 13, 69};
for (vector<int>::iterator i = numbers.begin(); i != numbers.end(); i++)
{
  *i /= 2;
}

// Equivalent code
for (int i = 0; i < numbers.size(); i++)
{
  numbers[i] /= 2;
}
```

Using Iterators (2)

- Example: Print each string element and its length

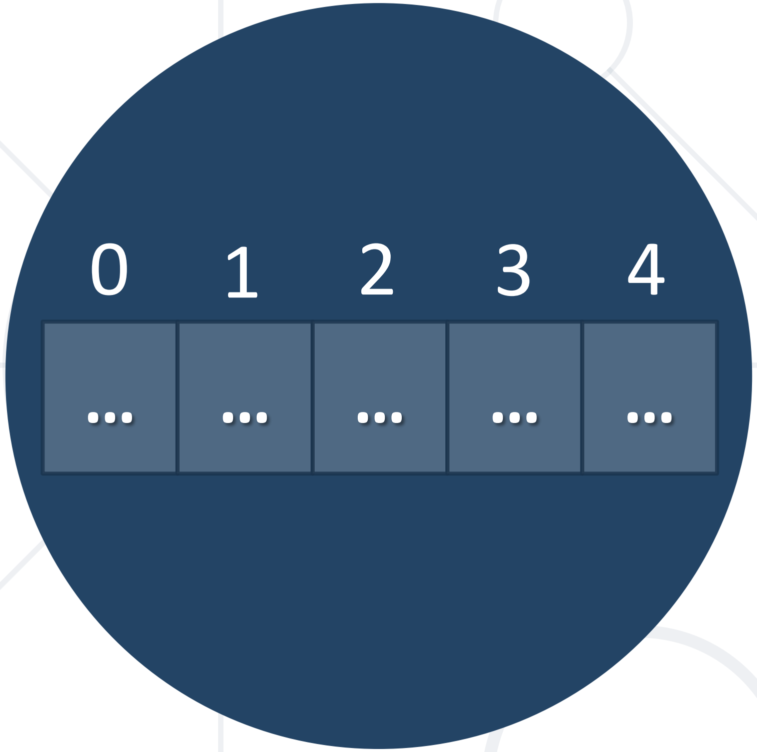
```
vector<string> words {"the", "quick", "purple", "fox"};
for (vector<string>::iterator i = words.begin(); i != words.end(); i++)
{
  cout << *i << ": " << i->size() << endl;
}

// Equivalent code
for (int i = 0; i < words.size(); i++)
{
  cout << words[i] << ": " << words[i].size() << endl;
}
```

Using Iterators with Vectors

LIVE DEMO

Why Use Iterators?

- Vectors may not need iterators, because they have indexes
  - They have sequential elements accessible by `operator[]`
- Not all containers have indexes
  - Only `std::array`, `std::vector` & `std::deque` have indexes
  - The other containers don't offer access by index
- Iterators work on all containers, abstract-away container details
  - No matter what container you iterate, code is the same



Lists

std::list

- Represents elements connected to each other in a sequence
  - `std::list<int> values; std::list<string> names;`
  - Each element connects to the previous and next element: *Like Christmas lights*
- All element access is done with iterators
- Can add or remove elements anywhere in `O(1)` time
  - Requires iterator to where an element should be **added** / **removed**
- `push_back()`, `push_front()`, `insert()`, `size()`

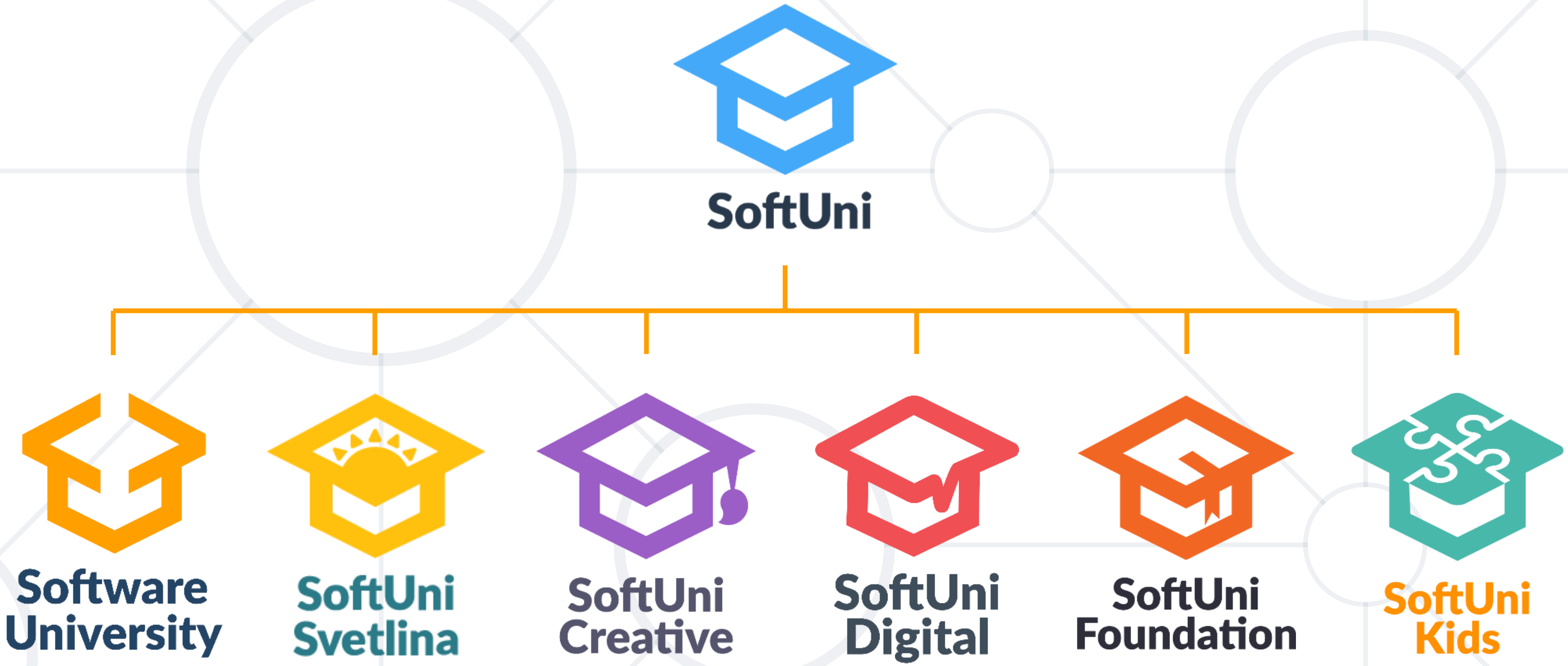
Lists

LIVE DEMO

Summary

- We usually measure performance based on input
  - We care how quickly much performance degrades based on input size
  - We use Big-O notation to denote that
- STL Vectors
- Iterators
- Lists

Questions?



SoftUni

Software University

SoftUni Svetlina

SoftUni Creative

SoftUni Digital

SoftUni Foundation

SoftUni Kids

SoftUni Diamond Partners

DRAFT KINGS

INDEAVR
Serving the high achievers

Coca-Cola HBC
Bulgaria

SCHWARZ

LIDL

Kaufland

Postbank
Решения за твоето утре

POKERSTARS | CAREERS

SCHWARZ Produktion

prezero

SUPER
HOSTING
.BG

SOFTWARE
GROUP

taulia®

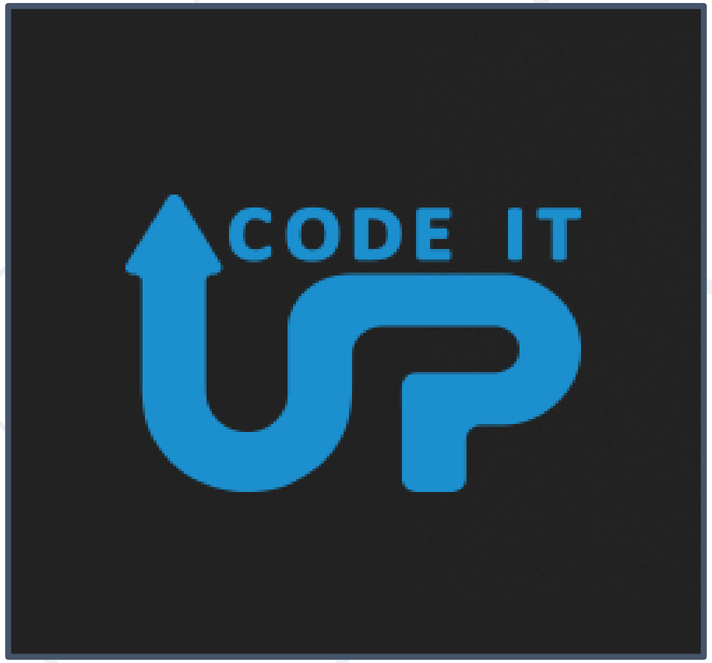
PHAR VISION

SmartIT

createX

MOTION SOFTWARE

Educational Partners



VIRTUAL RACING SCHOOL
VRS

License

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