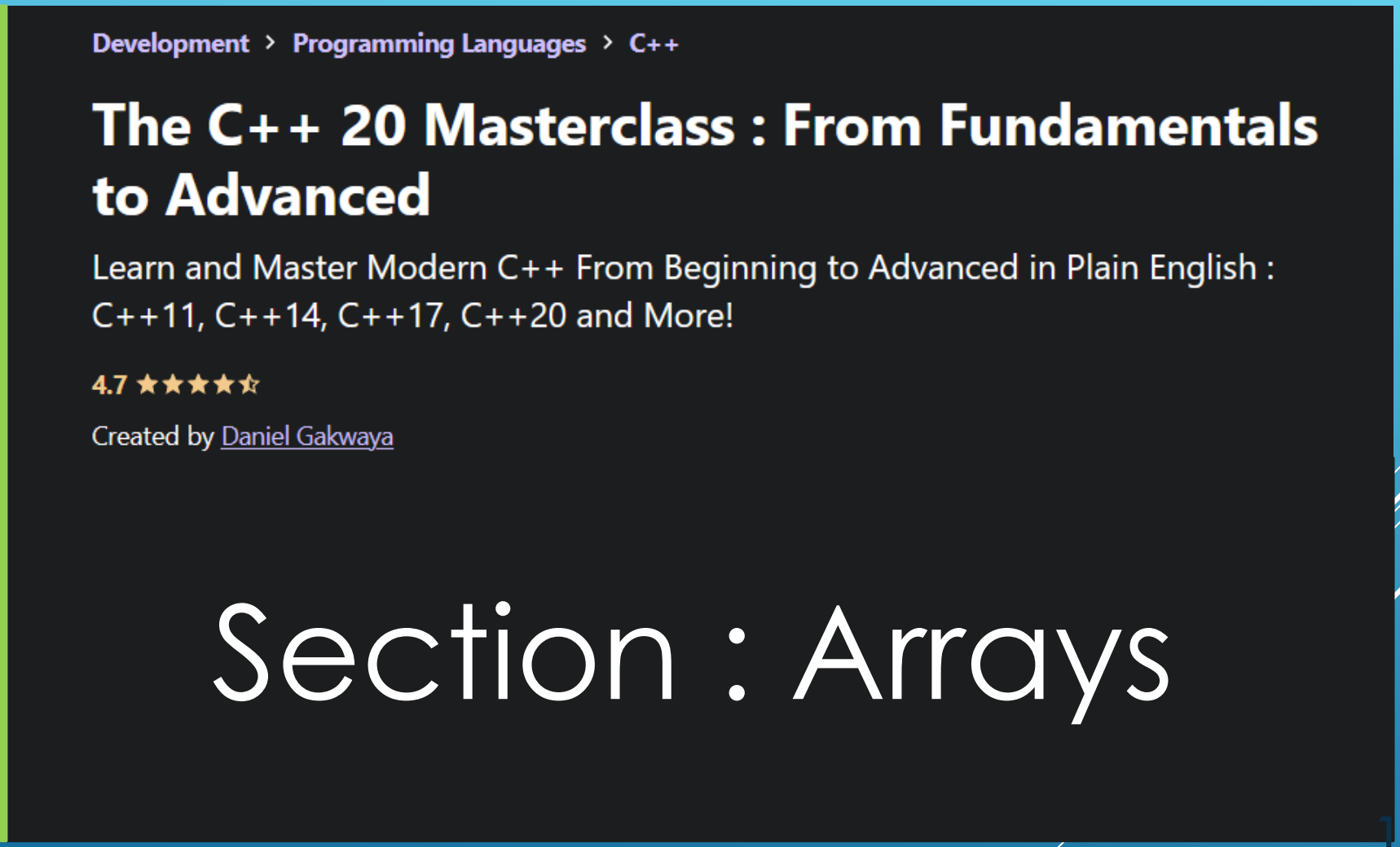
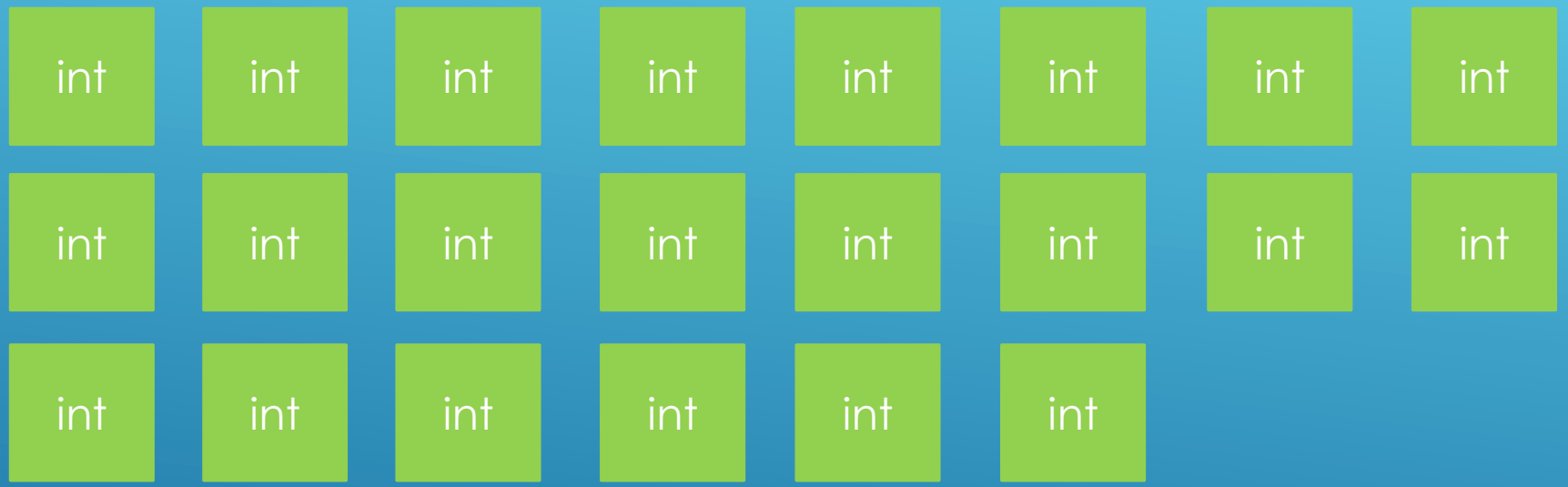
Slides

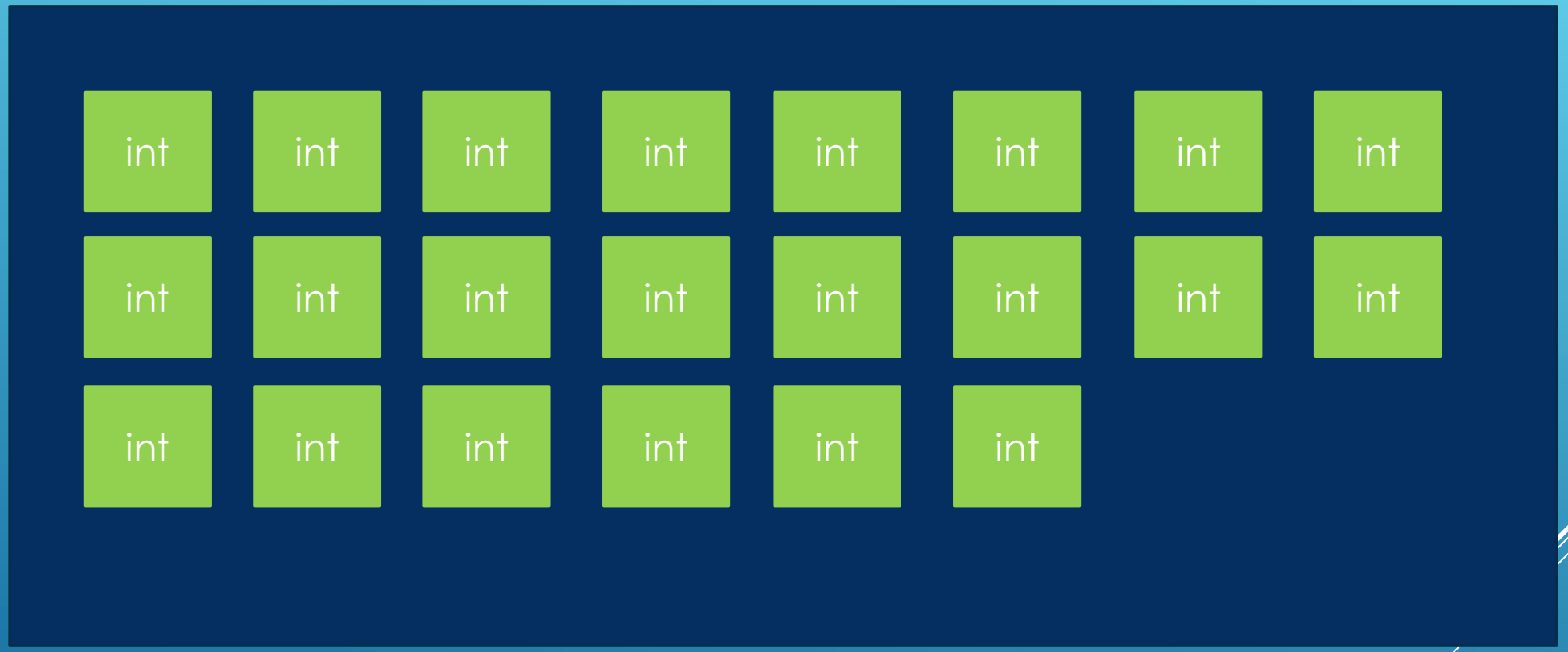


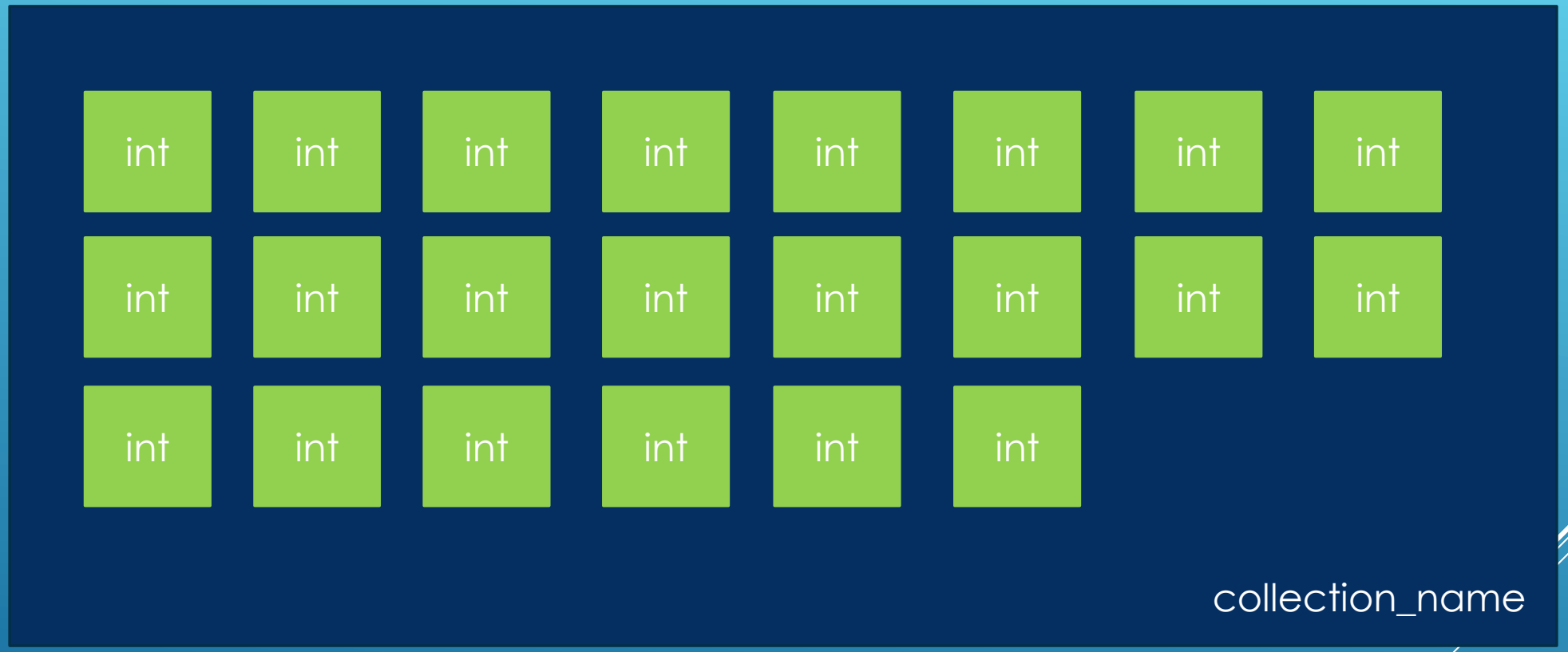
Section : Arrays

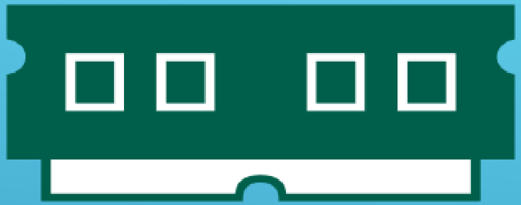
Slide intentionally left empty

Arrays : Introduction

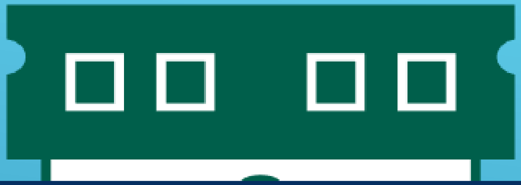

int int int int int int int int
int int int int int int int int
int int int int int int


int
int
int
int
int
int
int
int
int
int
int
int
int
int
int
int
int
int
int
int
int
int
int
int

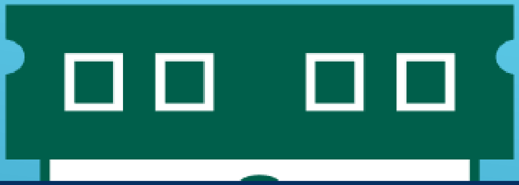

int
int
int
int
int
int
int
int
int
int
int
int
int
int
int
int
int
int
int
int
int
int
int
int
collection_name



0	1	1	0	1	1	1	0	0	0	1	1	0	0	0	0	1	1	1	1	0	0	0	1	0	0	1	1	1	1	1	1
0	1	1	0	1	1	1	0	0	0	1	1	0	0	0	0	1	1	1	1	0	0	0	1	0	0	1	1	1	1	1	1
0	1	1	0	1	1	1	0	0	0	1	1	0	0	0	0	1	1	1	1	0	0	0	1	0	0	1	1	1	1	1	1
0	1	1	0	1	1	1	0	0	0	1	1	0	0	0	0	1	1	1	1	0	0	0	1	0	0	1	1	1	1	1	1
0	1	1	0	1	1	1	0	0	0	1	1	0	0	0	0	1	1	1	1	0	0	0	1	0	0	1	1	1	1	1	1
0	1	1	0	1	1	1	0	0	0	1	1	0	0	0	0	1	1	1	1	0	0	0	1	0	0	1	1	1	1	1	1
0	1	1	0	1	1	1	0	0	0	1	1	0	0	0	0	1	1	1	1	0	0	0	1	0	0	1	1	1	1	1	1
0	1	1	0	1	1	1	0	0	0	1	1	0	0	0	0	1	1	1	1	0	0	0	1	0	0	1	1	1	1	1	1
0	1	1	0	1	1	1	0	0	0	1	1	0	0	0	0	1	1	1	1	0	0	0	1	0	0	1	1	1	1	1	1
0	1	1	0	1	1	1	0	0	0	1	1	0	0	0	0	1	1	1	1	0	0	0	1	0	0	1	1	1	1	1	1
0	1	1	0	1	1	1	0	0	0	1	1	0	0	0	0	1	1	1	1	0	0	0	1	0	0	1	1	1	1	1	1
0	1	1	0	1	1	1	0	0	0	1	1	0	0	0	0	1	1	1	1	0	0	0	1	0	0	1	1	1	1	1	1



0	1	1	0	1	1	1	0	0	0	1	1	0	0	0	0	1	1	1	1	0	0	0	1	0	0	1	1	1	1	1	1
0	1	1	0	1	1	1	0	0	0	1	1	0	0	0	0	1	1	1	1	0	0	0	1	0	0	1	1	1	1	1	1
0	1	1	0	1	1	1	0	0	0	1	1	0	0	0	0	1	1	1	1	0	0	0	1	0	0	1	1	1	1	1	1
0	1	1	0	1	1	1	0	0	0	1	1	0	0	0	0	1	1	1	1	0	0	0	1	0	0	1	1	1	1	1	1
0	1	1	0	1	1	1	0	0	0	1	1	0	0	0	0	1	1	1	1	0	0	0	1	0	0	1	1	1	1	1	1
0	1	1	0	1	1	1	0	0	0	1	1	0	0	0	0	1	1	1	1	0	0	0	1	0	0	1	1	1	1	1	1
0	1	1	0	1	1	1	0	0	0	1	1	0	0	0	0	1	1	1	1	0	0	0	1	0	0	1	1	1	1	1	1
0	1	1	0	1	1	1	0	0	0	1	1	0	0	0	0	1	1	1	1	0	0	0	1	0	0	1	1	1	1	1	1
0	1	1	0	1	1	1	0	0	0	1	1	0	0	0	0	1	1	1	1	0	0	0	1	0	0	1	1	1	1	1	1
0	1	1	0	1	1	1	0	0	0	1	1	0	0	0	0	1	1	1	1	0	0	0	1	0	0	1	1	1	1	1	1
0	1	1	0	1	1	1	0	0	0	1	1	0	0	0	0	1	1	1	1	0	0	0	1	0	0	1	1	1	1	1	1
0	1	1	0	1	1	1	0	0	0	1	1	0	0	0	0	1	1	1	1	0	0	0	1	0	0	1	1	1	1	1	1

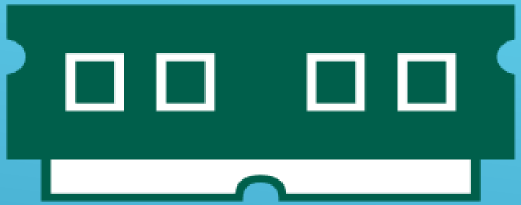


0	1	1	0	1	1	1	0	0	0	1	1	0	0	0	0	1	1	1	1	0	0	0	1	0	0	1	1	1	1	1	1
0	1	1	0	1	1	1	0	0	0	1	1	0	0	0	0	1	1	1	1	0	0	0	1	0	0	1	1	1	1	1	1
0	1	1	0	1	1	1	0	0	0	1	1	0	0	0	0	1	1	1	1	0	0	0	1	0	0	1	1	1	1	1	1
0	1	1	0	1	1	1	0	0	0	1	1	0	0	0	0	1	1	1	1	0	0	0	1	0	0	1	1	1	1	1	1
0	1	1	0	1	1	1	0	0	0	1	1	0	0	0	0	1	1	1	1	0	0	0	1	0	0	1	1	1	1	1	1
0	1	1	0	1	1	1	0	0	0	1	1	0	0	0	0	1	1	1	1	0	0	0	1	0	0	1	1	1	1	1	1
0	1	1	0	1	1	1	0	0	0	1	1	0	0	0	0	1	1	1	1	0	0	0	1	0	0	1	1	1	1	1	1
0	1	1	0	1	1	1	0	0	0	1	1	0	0	0	0	1	1	1	1	0	0	0	1	0	0	1	1	1	1	1	1
0	1	1	0	1	1	1	0	0	0	1	1	0	0	0	0	1	1	1	1	0	0	0	1	0	0	1	1	1	1	1	1
0	1	1	0	1	1	1	0	0	0	1	1	0	0	0	0	1	1	1	1	0	0	0	1	0	0	1	1	1	1	1	1
0	1	1	0	1	1	1	0	0	0	1	1	0	0	0	0	1	1	1	1	0	0	0	1	0	0	1	1	1	1	1	1
0	1	1	0	1	1	1	0	0	0	1	1	0	0	0	0	1	1	1	1	0	0	0	1	0	0	1	1	1	1	1	1

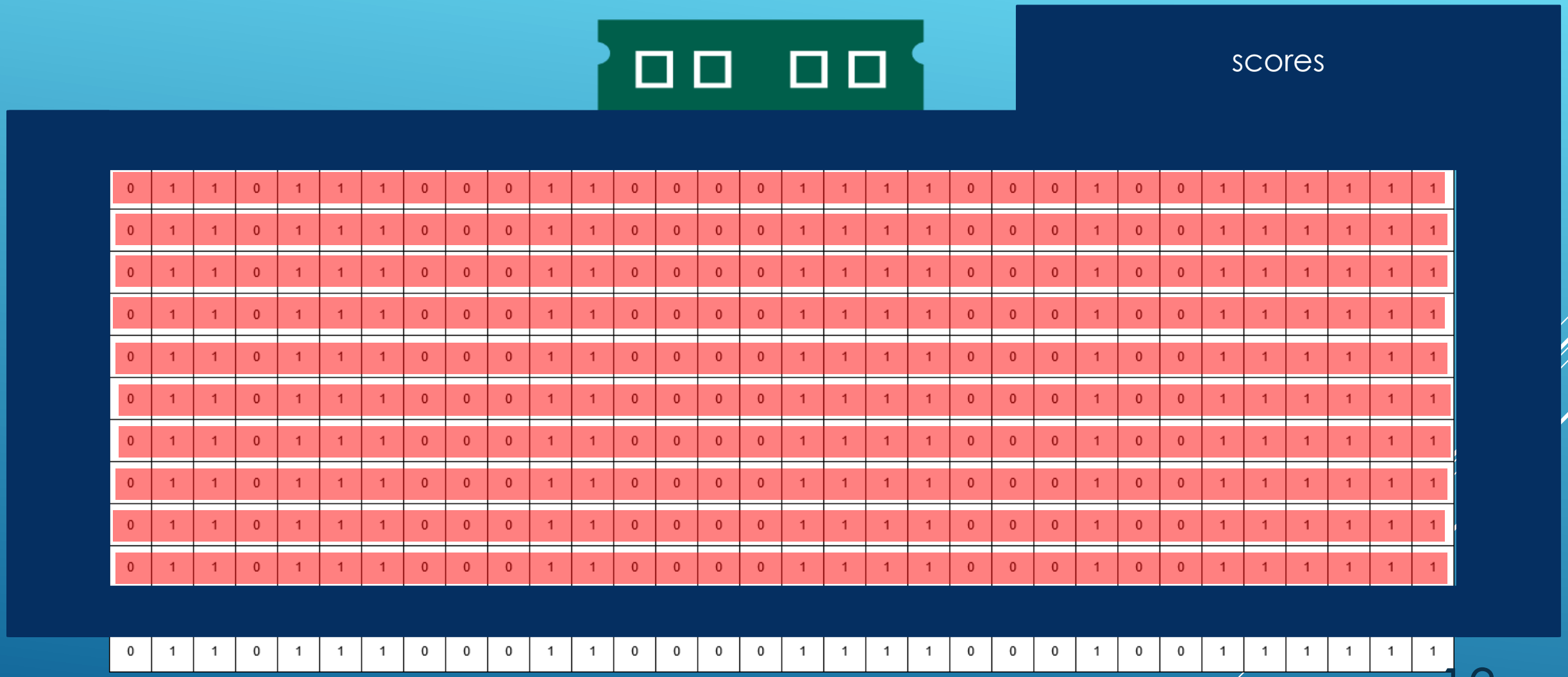
collection_name

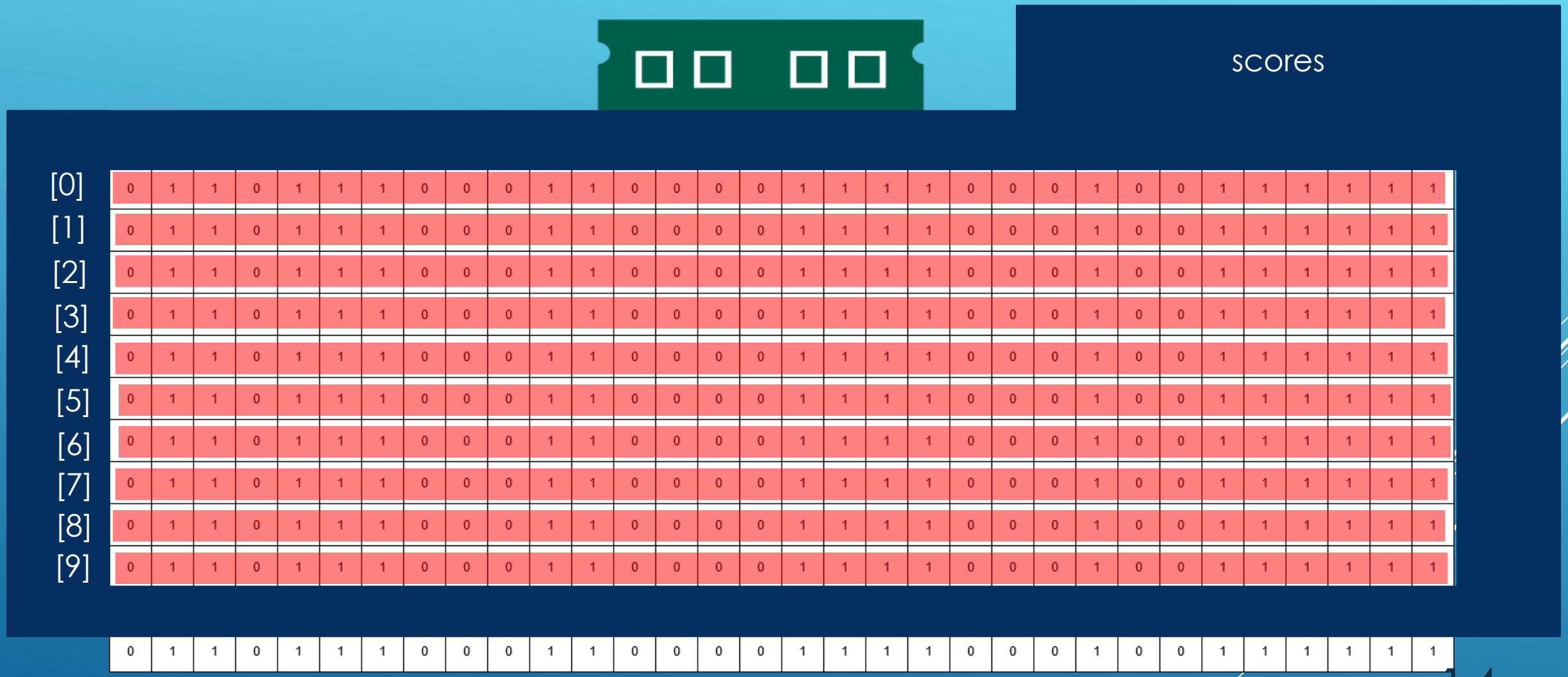
Slide intentionally left empty

Declaring and Using Arrays



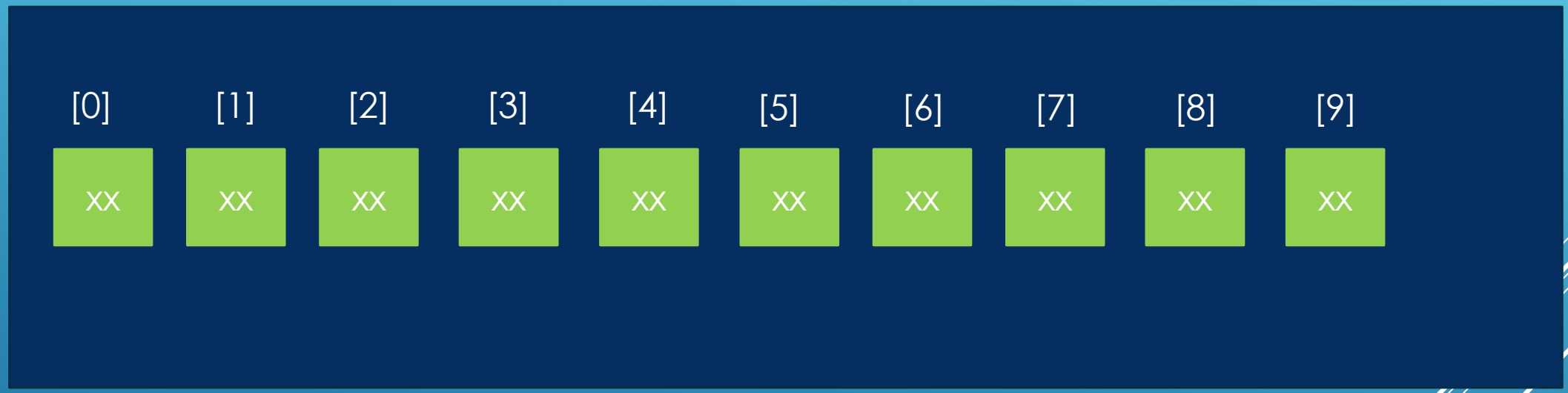
0	1	1	0	1	1	1	0	0	0	1	1	0	0	0	0	1	1	1	1	0	0	0	1	0	0	1	1	1	1	1	1
0	1	1	0	1	1	1	0	0	0	1	1	0	0	0	0	1	1	1	1	0	0	0	1	0	0	1	1	1	1	1	1
0	1	1	0	1	1	1	0	0	0	1	1	0	0	0	0	1	1	1	1	0	0	0	1	0	0	1	1	1	1	1	1
0	1	1	0	1	1	1	0	0	0	1	1	0	0	0	0	1	1	1	1	0	0	0	1	0	0	1	1	1	1	1	1
0	1	1	0	1	1	1	0	0	0	1	1	0	0	0	0	1	1	1	1	0	0	0	1	0	0	1	1	1	1	1	1
0	1	1	0	1	1	1	0	0	0	1	1	0	0	0	0	1	1	1	1	0	0	0	1	0	0	1	1	1	1	1	1
0	1	1	0	1	1	1	0	0	0	1	1	0	0	0	0	1	1	1	1	0	0	0	1	0	0	1	1	1	1	1	1
0	1	1	0	1	1	1	0	0	0	1	1	0	0	0	0	1	1	1	1	0	0	0	1	0	0	1	1	1	1	1	1
0	1	1	0	1	1	1	0	0	0	1	1	0	0	0	0	1	1	1	1	0	0	0	1	0	0	1	1	1	1	1	1
0	1	1	0	1	1	1	0	0	0	1	1	0	0	0	0	1	1	1	1	0	0	0	1	0	0	1	1	1	1	1	1
0	1	1	0	1	1	1	0	0	0	1	1	0	0	0	0	1	1	1	1	0	0	0	1	0	0	1	1	1	1	1	1
0	1	1	0	1	1	1	0	0	0	1	1	0	0	0	0	1	1	1	1	0	0	0	1	0	0	1	1	1	1	1	1


scores


scores
[0]
[1]
[2]
[3]
[4]
[5]
[6]
[7]
[8]
[9]
0 1 1 0 1 1 1 0 0 0 1 1 0 0 0 0 1 1 1 1 0 0 0 1 0 0 1 1 1 1 1 1

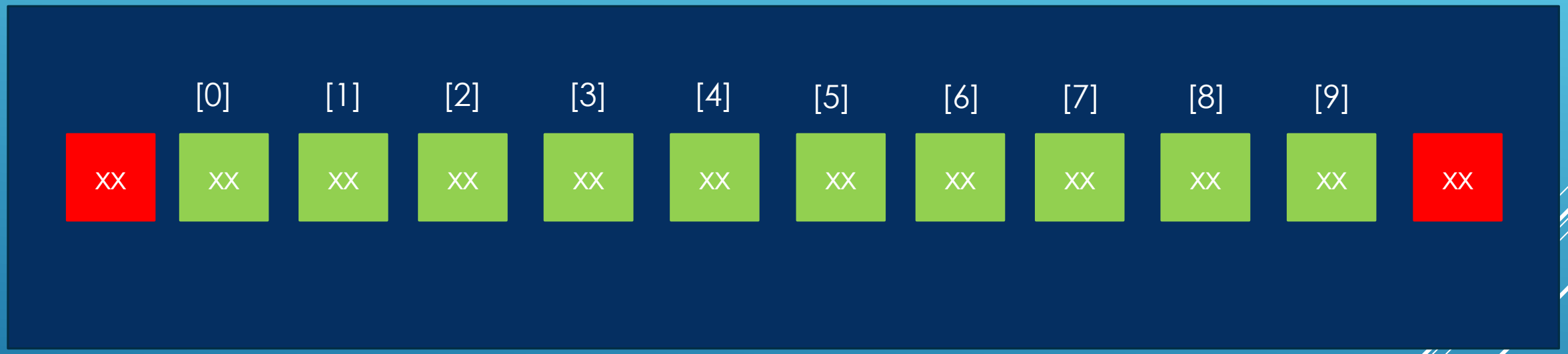
Declaring and Reading from an Array

```
//Declaring an array
int scores[10]; // An array storing 10 integers

//Reading values
std::cout << std::endl;
std::cout << "Reading out score values (manually) : " << std::endl;
std::cout << "scores[0] : " << scores[0] << std::endl;
std::cout << "scores[1] : " << scores[1] << std::endl;
//...
std::cout << "scores[9] : " << scores[9] << std::endl;
```


[0]
[1]
[2]
[3]
[4]
[5]
[6]
[7]
[8]
[9]
xx
xx
xx
xx
xx
xx
xx
xx
xx
xx

Array Bounds

```
//Reading past bounds of your array : BAD!
//It's going to read out something you didn't put there.
std::cout << "scores[10] : " << scores[10] << std::endl;
```

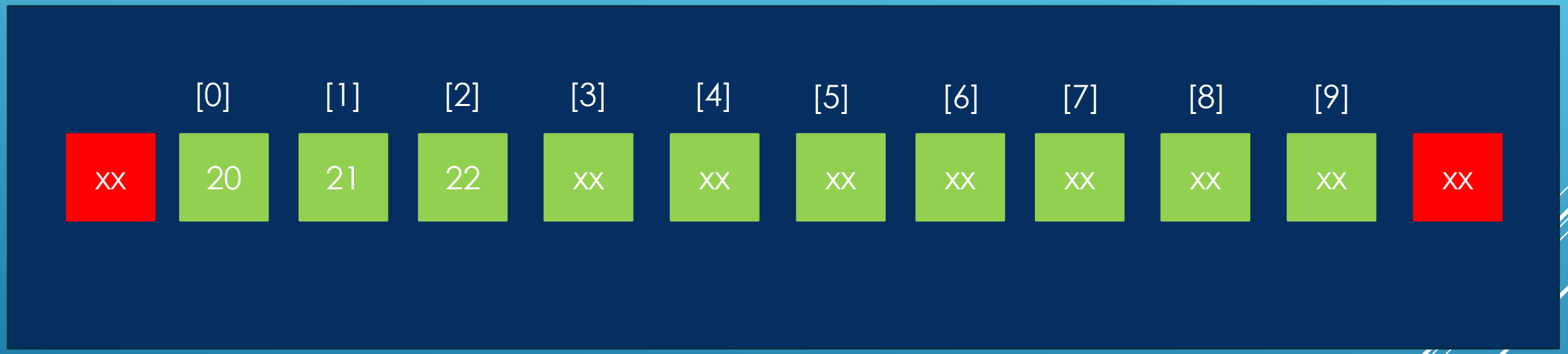

[0]
[1]
[2]
[3]
[4]
[5]
[6]
[7]
[8]
[9]
xx
xx
xx
xx
xx
xx
xx
xx
xx
xx
xx
xx

Looping through an array

```
//Can read through a loop

for( size_t i{0} ; i < 10 ; ++i){
    std::cout << "scores[" << i << "] : " << scores[i] << std::endl;
}
```

Writing data into an array

```
int scores[10]; // An array storing 10 integers

//Writing data in an array
scores[0] = 20;
scores[1] = 21;
scores[2] = 22;

//Writting out of bounds. BAD!
scores[22] = 300;

std::cout << std::endl;
std::cout << "Manually writing data in array : " << std::endl;
for( size_t i{0} ; i < 10 ; ++i){
    std::cout << "scores[" << i << "] : " << scores[i] << std::endl;
}
```


[0]
[1]
[2]
[3]
[4]
[5]
[6]
[7]
[8]
[9]
xx
20
21
22
xx
xx
xx
xx
xx
xx
xx
xx

Writing data with a loop

```cpp
int scores[10]; // An array storing 10 integers

std::cout << std::endl;
std::cout << "Writing data in array with loop : " << std::endl;

//Write data
for( size_t i{0} ; i < 10 ; ++i){
    scores[i] = i *3;
}

//Read data out
for( size_t i{0} ; i < 10 ; ++i){
    std::cout << "scores[" << i << "] : " << scores[i] << std::endl;
}
```

Initialize the array at declaration

```
//Declare and initialize at the same time
std::cout << std::endl;
std::cout << "Declare and initialize at the same time : " << std::endl;

double salaries[5] {12.7, 7.5, 13.2, 8.1, 9.3};

for(size_t i{0}; i < 5; ++i){
    std::cout << "salary[" << i << "] : " << salaries[i] << std::endl;
}
```

Array Initialization : Omitting elements

```
//If you don't initialize all the elements, those you leave out
//are initialized to 0

std::cout << std::endl;
std::cout << "Leaving out some elements un-initialized : " << std::endl;

int families[5] {12, 7, 5};

for(size_t i{0}; i < 5; ++i){
    std::cout << "families[" << i << "] : " << families[i] << std::endl;
}
```

Array Declaration : Omit size

```
//Omit the size of the array at declaration
int class_sizes[] {10,12,15,11,18,17};

for(auto value : class_sizes){
    std::cout << "value : " << value << std::endl;
}
```

Constant Arrays

```
//Constant arrays , can't be modified.
const int multipliers [] { 22,30,15};
multipliers[1] = 20; // Can't change elements of a const array: ERROR
```

Operations on data stored in arrays

```
//Sum up scores array, store result in sum
int sum {0};

for( int element : scores){
    sum += element;
}
std::cout << "Score sum : " << sum << std::endl;
```

Slide intentionally left empty

Size of An Array

std::size() [C++17]

```
int scores[] {10,12,15,11,18,17,22,23,24};

//Can get the size with std::size
std::cout << "scores size : " << std::size(scores) << std::endl;

//Print data out
for( size_t i{0} ; i < std::size(scores) ; ++i){
    std::cout << "scores[" << i << "] : " << scores[i] << std::endl;
}
```

Array size with sizeof

```
std::cout << "size of scores : " << sizeof(scores) << std::endl;
std::cout << "size of scores[0] : " << sizeof(scores[0]) << std::endl;
std::cout << "score item count : " << (sizeof(scores)/ sizeof(scores[0])) << std::endl;

size_t count {sizeof(scores)/ sizeof(scores[0])};

std::cout << "Printing out array items : " << std::endl;
for( size_t i{0} ; i < count ; ++i){
    std::cout << "scores[" << i << "] : " << scores[i] << std::endl;
}
```

Looping through an array

```
int scores[] {10,12,15,11,18,17,22,23,24};

std::cout << "Using plain old range based for loop " << std::endl;

for ( auto score : scores){
    std::cout << "score : " << score << std::endl;
}
```

Slide intentionally left empty

Arrays of Characters

Array of characters

```
//Declare array
char message [5]  {'H','e','l','l','o'};

//Print out the array through looping
std::cout << "message : ";
for( auto c : message){
    std::cout << c ;
}
std::cout << std::endl;
std::cout << "size : " << std::size(message) <<std::endl;


//Can also modify elements of the char array

std::cout << std::endl;
std::cout << "Modify array data : " << std::endl;

message[1] = 'a';

//Print out the array
std::cout << "message : ";
for( auto c : message){
    std::cout << c ;
}
```

Direct print out

```
char message [5]  {'H','e','l','l','o'};
std::cout << "message : " << message << std::endl;
```

Null termination

```
char message [5]  {'H','e','l','l','o', \0 };
std::cout << "message : " << message << std::endl;
```

Proper null termination

```
char message1[] {'H','e','l','l','o','\0'};

std::cout << "message1 : " << message1 << std::endl;
std::cout <<"size : " << std::size(message1) << std::endl;
```

Auto filled in null characters

```
char message2[6] {'H','e','l','l','o'};

std::cout << "message2 : " << message2 << std::endl;
std::cout << "size : "<< std::size(message2) << std::endl;
```

Looks may be deceiving

```
char message3 [] {'H','e','l','l','o'}; // This is not a c string ,
                                        //as there is not null character
std::cout << "message3: " << message3 << std::endl;
std::cout << "size : " << std::size(message3) << std::endl;// Will probably print some
                                                           //garbage after our hello message
```

Literal C-strings

```
//Can also define a literal C string
std::cout << std::endl;
char message4 [] {"Hello"}; // An implicit '\0' character is appended to the
                            // end of the string, making it a c string
std::cout << "message4 : " << message4 << std::endl;
std::cout << "size : " << std::size(message4) << std::endl;


//Can even have spaces between characters
std::cout << std::endl;
char message5[] {"Hello World!" };
std::cout << "message5 : " << message5 << std::endl;
std::cout << "size : " << std::size(message5) << std::endl;
```

Arrays of char are special

```
//Can't direct print arrays other than that of chars
std::cout << std::endl;
int numbers[] {1,2,3,4,5,6,7,8,9,0};
std::cout << "numbers : " << numbers << std::endl;
```

Slide intentionally left empty

Bounds of an Array

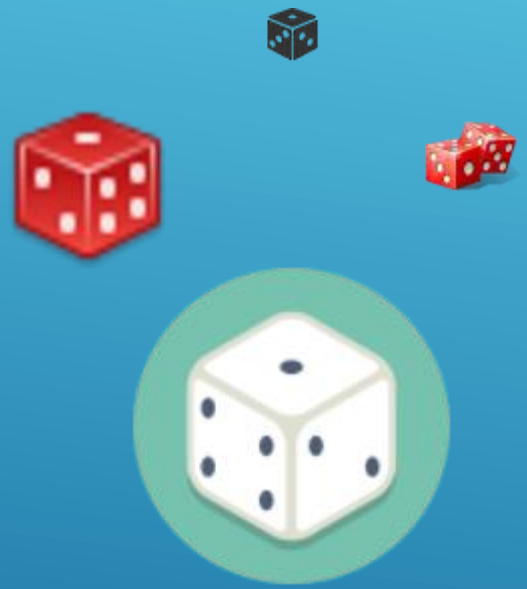
Array bounds

```
int numbers[] {1,2,3,4,5,6,7,8,9,0};

//Read beyond bounds : Will read garbage or crash your program
std::cout << "numbers[12] : " << numbers[12] << std::endl;

//Write beyond bounds. The compiler allows it. But you don't own
//the memory at index 12, so other programs may modify it and your
//program may read bogus data at a later time. Or you can even
//corrupt data used by other parts of your ptogram
numbers[12] = 1000;
std::cout << "numbers[12] : " << numbers[12] << std::endl;
```

Slide intentionally left empty

Generating Random Numbers

A lot of applications for random numbers



std::rand()

```
//srand
int random_num = std::rand(); // generates a number between 0 and RAND_MAX
std::cout << "number : " <<  random_num << std::endl;

random_num = std::rand();
std::cout << "number : " << random_num << std::endl;

std::cout << "RAND_MAX : " << RAND_MAX << std::endl; // 32767
```

Random ranges

```
//Range [0~10] : what you % with controls the upper bound

int random_num = std::rand() % 11; // Will be between [0~10]
std::cout << "number : " << random_num << std::endl;

for (size_t i{}; i < 30; ++i) {
    random_num = std::rand() % 11; // Will be between [0~10]
    std::cout << "number : " << random_num << std::endl;
}
```

Random ranges

```
//Range [1~15]
int random_num = (std::rand() % 15 )+ 1; // [1~15]

for (size_t i{}; i < 30; ++i) {
    random_num = (std::rand() % 15) + 1; // Will be between [1~15]
    std::cout << "number : " << random_num << std::endl;
}
```

std::rand	Same sequence each time the program runs

Seed

```
std::time(0) // Time since January 1st, 1970 at 00:00:00 AM
std::srand(std::time(0)); // Seed
```

Proper seeding

```
std::srand(std::time(0)); // Seed

//Range [1~15]
int random_num = (std::rand() % 15 )+ 1; // [1~15]

for (size_t i{}; i < 30; ++i) {
    random_num = (std::rand() % 15) + 1; // Will be between [1~15]
    std::cout << "number : " << random_num << std::endl;
}
```

Slide intentionally left empty

Fortune Teller v1

Fortune Teller : General Flow Control

```
bool end{false};

int  max_length{15};
char name [max_length]{};

std::cout << "What's your name dear :" << std::endl;

std::cin.getline(name, max_length);

while(!end){

    std::cout << "Do you want me to try again ? (Y | N) : ";

    char go_on;
    std::cin >> go_on;

    end = ((go_on == 'Y') || (go_on == 'y')) ? false : true;

}
```

Storing the predictions

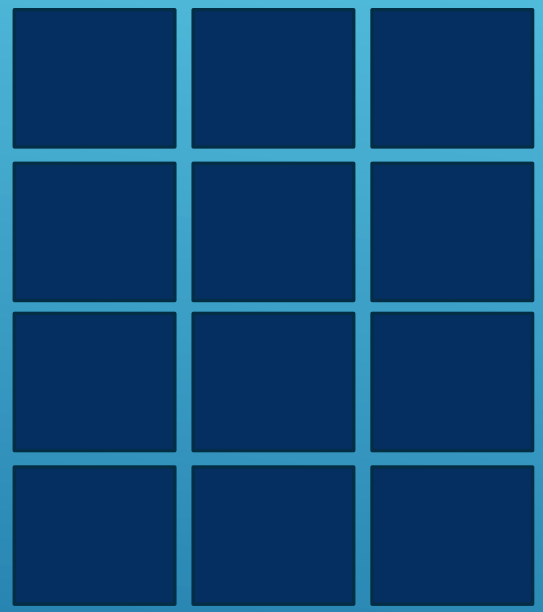
```
char prediction0[]  {"a lot of kinds running in the backyard!"};
char prediction1[]  {"a lot of empty beer bootles on your work table."};
char prediction2[]  {"you Partying too much with kids wearing weird clothes."};
char prediction3[]  {"you running away from something really scary"};
char prediction4[]  {"clouds gathering in the sky and an army standing ready for war"};
char prediction5[]  {"dogs running around in a deserted empty city"};
char prediction6[]  {"a lot of cars stuck in a terrible traffic jam"};
char prediction7[]  {"you sitting in the dark typing lots of lines of code on your dirty computer"};
char prediction8[]  {"you yelling at your boss. And oh no! You get fired!"};
char prediction9[]  {"you laughing your lungs out. I've never seen this before."};
```

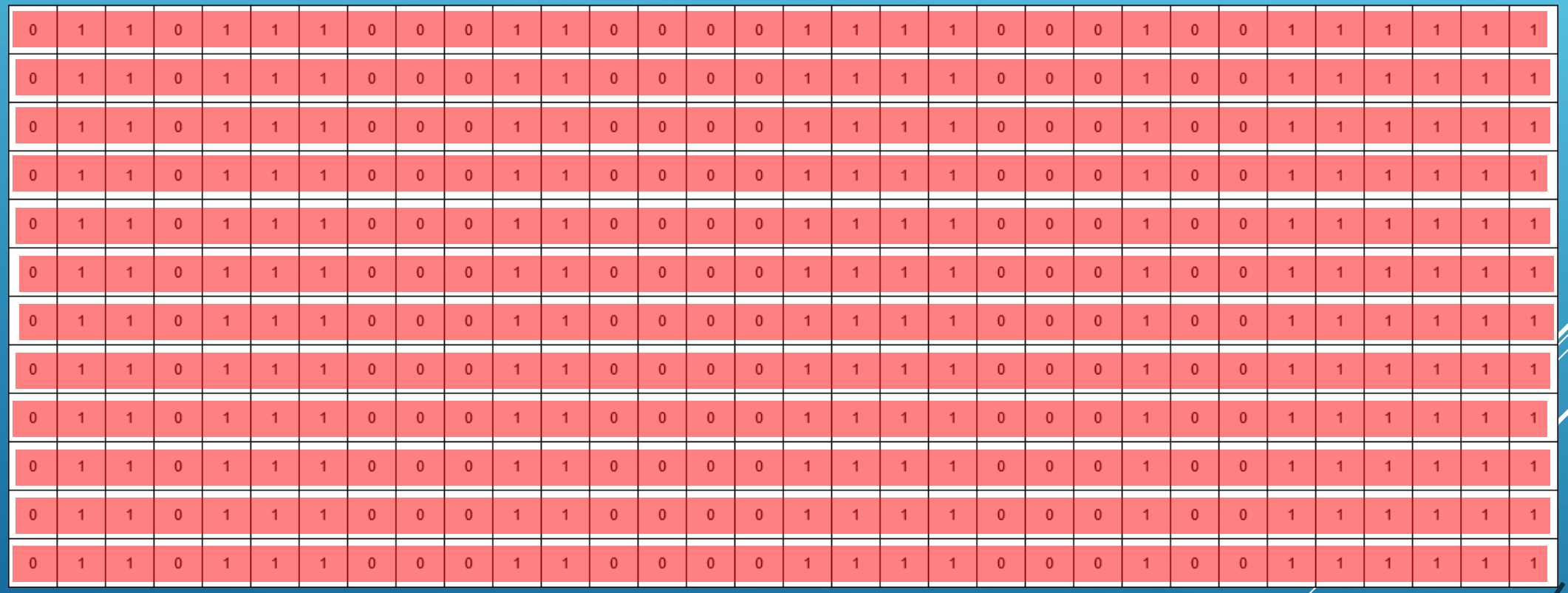
Choosing the prediction randomly

```
while(!end){
    std::cout << "Oh dear " <<name << ", I see ";

    switch (static_cast<size_t>((std::rand() % count) + starting_point)){
        case 0 :
          std::cout << prediction0 << std::endl;
          break;
        case 1 :
          std::cout << prediction1 << std::endl;
          break;
          /* ...
        case 9 :
          std::cout << prediction9 << std::endl;
          break;
        default :
          std::cout << ", huum, I don't see anything" <<std::endl;
    }
    std::cout << "Do you want me to try again ? (Y | N) : ";

    char go_on;
    std::cin >> go_on;

    end = ((go_on == 'Y') || (go_on == 'y')) ? false : true;
}
```

Slide intentionally left empty

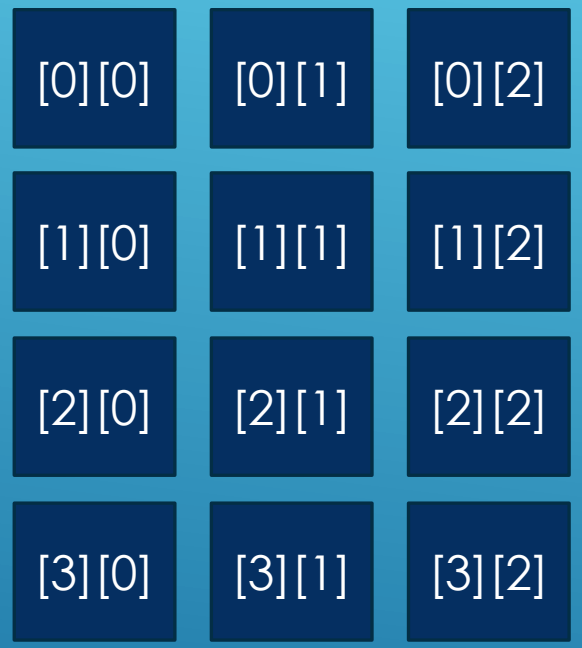
Multi Dimensional Arrays

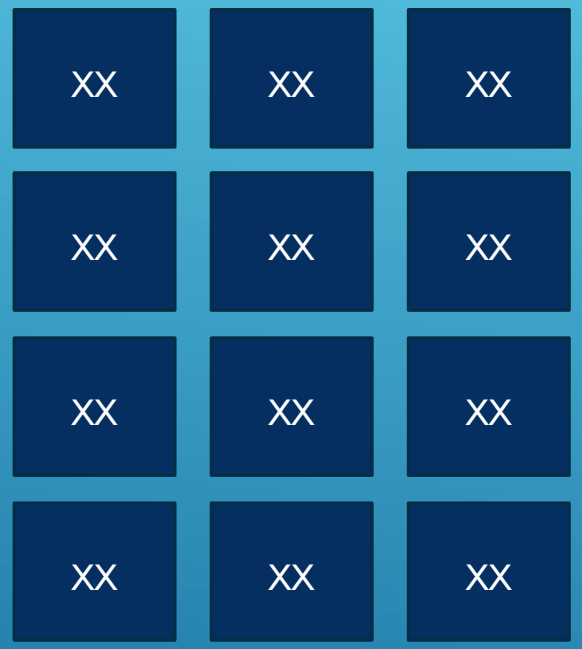



0 1 1 0 1 1 1 0 0 0 1 1 0 0 0 0 1 1 1 1 0 0 0 1 0 0 1 1 1 1 1 1

Declaring a 2d array and printing it out

```
//Declaring a 2d int array
int packages [4][3]; // Occupies 48 bytes in memory.

std::cout << "Size of packages :" << sizeof(packages) << std::endl;// Occupied memory

//Element count in packages will be 4. Because packages contains 4 rows and 3
//columns, we can say that packages contains 4 items, each of which contains
//three items.
std::cout << "element count in packages :" << std::size(packages) << std::endl;



//The 2d array contains garbage data by default, lets print it out
//Using hardcoded values 4 and 3 ( magic numbers)
std::cout << std::endl;
std::cout << "Print out un-initialized array using magic numbers for dimensions : " << std::endl;
for ( size_t i{0}; i < 4 ; ++i){
    for( size_t j{0} ; j < 3 ; ++j){
        std::cout << "Item (" << i << "," << j << ") : " << packages[i][j] << std::endl;
    }
}
```


[0][0]
[0][1]
[0][2]
[1][0]
[1][1]
[1][2]
[2][0]
[2][1]
[2][2]
[3][0]
[3][1]
[3][2]


XX XX XX
XX XX XX
XX XX XX
XX XX XX

Printing data out

```
//Printig out with std::size for more flexibility
std::cout <<std::endl;
std::cout << "Print out un-initialized array by dynamicaly querying the dimensions : " << std::endl;
std::cout << "Printing out elements in the packages array : " << std::endl;
for (size_t i{0} ; i < std::size(packages) ; ++i){
    for( size_t j{0} ; j < std::size(packages[i]) ; ++j){
        std::cout << "Item (" << i << "," << j << ") : " << packages[i][j] << std::endl;
    }
}
```

Initialize multi dimensional array with data

```
//Initializing the array with data
int packages1 [4] [3] {
    {1,2,3},
    {4,5,6},
    {7,8,9},
    {10,11,12}
};

//If we print it , we'll see the numbers
//Print packages1
std::cout << std::endl;
std::cout << "Printing out initialized 2d array packages1 : " << std::endl;
for (size_t i{0} ; i < std::size(packages1) ; ++i){
    for( size_t j{0} ; j < std::size(packages1[i]) ; ++j){
        std::cout << "Item (" << i << "," << j << ") : " << packages1[i][j] << std::endl;
    }
}
```

Declaring a 3d array

```
//3D arrays are defined in the same way. We just use three sets of indexes
// 3 lights per room, 5 rooms per house 7 houses per block
int house_block [7] [5] [3] {
    {
        {1,2,3},{4,5,6},{7,8,9},{10,11,12},{13,14,15}
    },
    {
        {16,17,18},{19,20,21},{22,23,24},{25,26,27},{28,29,30}
    },
    {
        { 31,32,33},{34,35,36},{37,38,39},{40,41,42},{43,44,45}
    },
    {
         {46,47,48},{49,50,51},{52,53,54},{55,56,57},{58,59,60}
    },
    {
         {61,62,63},{64,65,66},{67,68,69},{70,71,72},{73,74,75}
    },
    {
        {76,77,78},{79,80,81},{82,83,84},{85,86,87},{88,89,90}
    },
    {
         {91,92,93},{94,95,96},{97,98,99},{100,101,102},{103,104,105}
    }
};
```

Printing out a 3d array

```
std::cout << std::endl;
std::cout << "Printing out 3d house_block array : " << std::endl;

for (size_t i {0} ; i < std::size(house_block ); ++i){

    for( size_t j{0}; j < std::size(house_block[i]) ; ++j){

        std::cout << "[";
        for( size_t k{0}; k < std::size(house_block[i][j]) ; ++k){

            std::cout << house_block[i][j][k] << " ";
        }
        std::cout <<"] " ;//Separate elements for good visualization
    }
    std::cout << std::endl;//Separate elements for good visualization
}
```

Omitting dimensions out

```
//You can omit the number of elements inside an [] when declaring a multi dimensional
// array but only one, and it has to be the left most.
// int packages2 [] [num_cols] will work, but int packages2 [] [] won't work and give a
//compile error

//const size_t num_rows{4};
const size_t num_cols{3};

int packages2 [] [num_cols] {
    {1,2,3},
    {4,5,6},
    {7,8,9},
    {10,11,12},
    {100,110,120} // Can add as many triplets in packages2 as we want
};

std::cout << std::endl;
std::cout << "Ommiting leftmost dimension for 2d array : " << std::endl;
for (size_t i{0} ; i < std::size(packages2) ; ++i){
    for( size_t j{0} ; j < std::size(packages2[i]) ; ++j){
        std::cout << "Item (" << i << "," << j << ") : " << packages2[i][j] << std::endl;
    }
}
```

Omitting dimensions out

```
//For 3d and really any multi dimensional array , you have to specify
//the number of elements in []'s , only the left most is not mandatory.
//Below is the example for 3D reproduced.Omitting the 5 or the 3 or both
//will cause a compile error.

int house_block1 [] [5] [3] {

    {
        {1,2,3},{4,5,6},{7,8,9},{10,11,12},{13,14,15}
    },
    {
        {16,17,18},{19,20,21},{22,23,24},{25,26,27},{28,29,30}
    },
    {
        { 31,32,33},{34,35,36},{37,38,39},{40,41,42},{43,44,45}
    },
    {
         {46,47,48},{49,50,51},{52,53,54},{55,56,57},{58,59,60}
    }

};
```

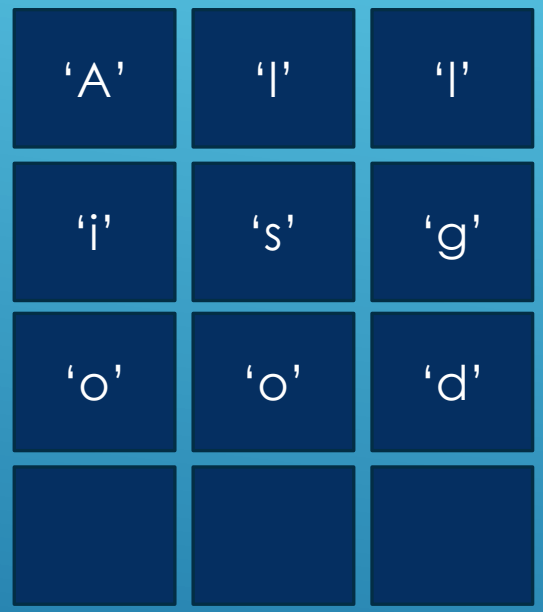
Auto filling

```
int house_block2 [] [5] [4] {

    {
        {1,2,3},{4},{7,8,9},{10,11,12},{13,14,15}  //The one element array will be autofilled
                                                   // with zeros to complete 4 elements
    },
    {
        {16,17,18},{19,20,21},{22,23,24},{25,26,27},{28,29,30}
    },
    {
        { 31,32,33},{34,35,36},{37,38,39},{40,41,42},{43,44,45}
    },
    {
         {46,47,48},{49,50,51},{52,53,54},{55,56,57}//,{58,59,60}  // The spots for {58,59,60
                                                                   // Will be autofilled with 0
    }

};
```

```
//Modifying elements
house_block2[1][2][2] = 1001;
```

Slide intentionally left empty

Multi Dimensional Arrays of Characters


'A'
'l'
'l'
'i'
's'
'g'
'o'
'o'
'd'

Declaration

```
const size_t name_length{15};
char members [][name_length] {
    {'J','o','h','n'},
    {'S','a','m','u','e','l',},
    {'R','a','s','h','i','d'},
    {'R','o','d','r','i','g','e','z'}
};

//Printing out like this is unsafe : may go over and print
//outside your valid memory block
//until a terminating null character is encountered.
std::cout << "Unsafe printing of members : " << std::endl;
for (size_t i {0}; i < std::size(members) ; ++i){
    std::cout << members[i] << std::endl;
}
```

Print out 2d array of char

```
//Can loop around manually printing out each character

std::cout << std::endl;
std::cout << "Printing out character by character manually : " << std::endl;
for (size_t i{0} ; i < std::size(members) ; ++i){

    for (size_t j{0} ; j < std::size(members[i]) ; ++j){

        std::cout << members[i][j] ;
    }
    std::cout << std::endl;

}
```

C-string literals

```
//Better : Using C-string litterals
//Compared to initialization with charactes with in '', this
// is even easier to type. The entire string is a single entity
//you can manage easily.
char members1 [][name_length] {
    "John",
    "Samuel",
    "Rashid",
    "Rodriguez"
};

//Printing out members1
std::cout << "Printing out members1 (C-string literals) : " << std::endl;
for (size_t i {0}; i < std::size(members1) ; ++i){
    std::cout << members1[i] << std::endl;
}
```

Predictions : Fortune Teller V1

```
char prediction0[]  {"a lot of kinds running in the backyard!"};
char prediction1[]  {"a lot of empty beer bootles on your work table."};
char prediction2[]  {"you Partying too much with kids wearing weird clothes."};
char prediction3[]  {"you running away from something really scary"};
char prediction4[]  {"clouds gathering in the sky and an army standing ready for war"};
char prediction5[]  {"dogs running around in a deserted empty city"};
char prediction6[]  {"a lot of cars stuck in a terrible traffic jam"};
char prediction7[]  {"you sitting in the dark typing lots of lines of code on your dirty computer"};
char prediction8[]  {"you yelling at your boss. And oh no! You get fired!"};
char prediction9[]  {"you laughing your lungs out. I've never seen this before."};
```

Predictions : Fortune Teller V2

```
//Updating our prediction declaration in the fortune teller game
char predictions [] [90] {
    "a lot of kinds running in the backyard!",
    "a lot of empty beer bootles on your work table.",
    "you Partying too much with kids wearing weird clothes.",
    "you running away from something really scary",
    "clouds gathering in the sky and an army standing ready for war",
    "dogs running around in a deserted empty city",
    "a lot of cars stuck in a terrible traffic jam",
    "you sitting in the dark typing lots of lines of code on your dirty computer",
    "you yelling at your boss. And oh no! You get fired!",
    "you laughing your lungs out. I've never seen this before."

};
```

Slide intentionally left empty

Fortune Teller v2

Fortune Teller : General Flow Control

```
bool end{false};

int  max_length{15};
char name [max_length]{};

std::cout << "What's your name dear :" << std::endl;

std::cin.getline(name, max_length);

while(!end){

    std::cout << "Do you want me to try again ? (Y | N) : ";

    char go_on;
    std::cin >> go_on;

    end = ((go_on == 'Y') || (go_on == 'y')) ? false : true;

}
```

Storing the predictions

```
char prediction0[]  {"a lot of kinds running in the backyard!"};
char prediction1[]  {"a lot of empty beer bootles on your work table."};
char prediction2[]  {"you Partying too much with kids wearing weird clothes."};
char prediction3[]  {"you running away from something really scary"};
char prediction4[]  {"clouds gathering in the sky and an army standing ready for war"};
char prediction5[]  {"dogs running around in a deserted empty city"};
char prediction6[]  {"a lot of cars stuck in a terrible traffic jam"};
char prediction7[]  {"you sitting in the dark typing lots of lines of code on your dirty computer"};
char prediction8[]  {"you yelling at your boss. And oh no! You get fired!"};
char prediction9[]  {"you laughing your lungs out. I've never seen this before."};
```

Choosing the prediction randomly

```
while(!end){
    std::cout << "Oh dear " <<name << ", I see ";

    switch (static_cast<size_t>((std::rand() % count) + starting_point)){
        case 0 :
          std::cout << prediction0 << std::endl;
          break;
        case 1 :
          std::cout << prediction1 << std::endl;
          break;
          /* ...
        case 9 :
          std::cout << prediction9 << std::endl;
          break;
        default :
          std::cout << ", huum, I don't see anything" <<std::endl;
    }
    std::cout << "Do you want me to try again ? (Y | N) : ";

    char go_on;
    std::cin >> go_on;

    end = ((go_on == 'Y') || (go_on == 'y')) ? false : true;
}
```

Updating string storage

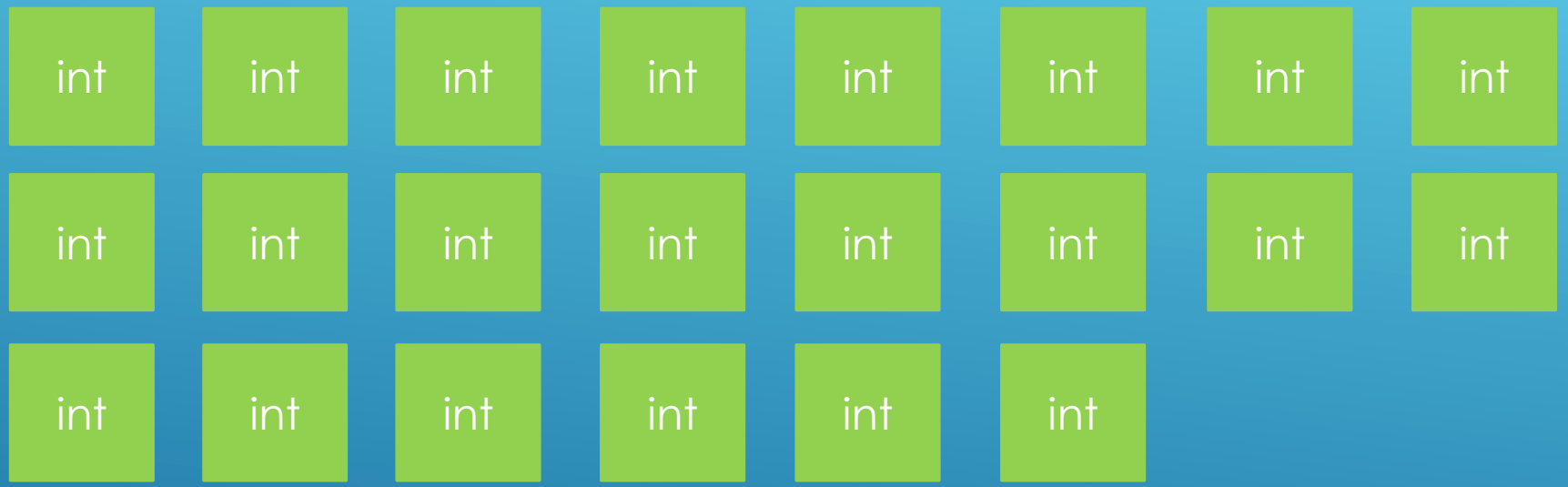
```
/*
char prediction0[]  {"a lot of kinds running in the backyard!"};
char prediction1[]  {"a lot of empty beer bootles on your work table."};
char prediction2[]  {"you Partying too much with kids wearing weird clothes."};
char prediction3[]  {"you running away from something really scary"};
char prediction4[]  {"clouds gathering in the sky and an army standing ready for war"};
char prediction5[]  {"dogs running around in a deserted empty city"};
char prediction6[]  {"a lot of cars stuck in a terrible traffic jam"};
char prediction7[]  {"you sitting in the dark typing lots of lines of code on your dirty computer"};
char prediction8[]  {"you yelling at your boss. And oh no! You get fired!"};
char prediction9[]  {"you laughing your lungs out. I've never seen this before."};
*/

char predictions [] [90] {
    "a lot of kinds running in the backyard!",
    "a lot of empty beer bootles on your work table.",
    "you Partying too much with kids wearing weird clothes.",
    "you running away from something really scary",
    "clouds gathering in the sky and an army standing ready for war",
    "dogs running around in a deserted empty city",
    "a lot of cars stuck in a terrible traffic jam",
    "you sitting in the dark typing lots of lines of code on your dirty computer",
    "you yelling at your boss. And oh no! You get fired!",
    "you laughing your lungs out. I've never seen this before."
};
```

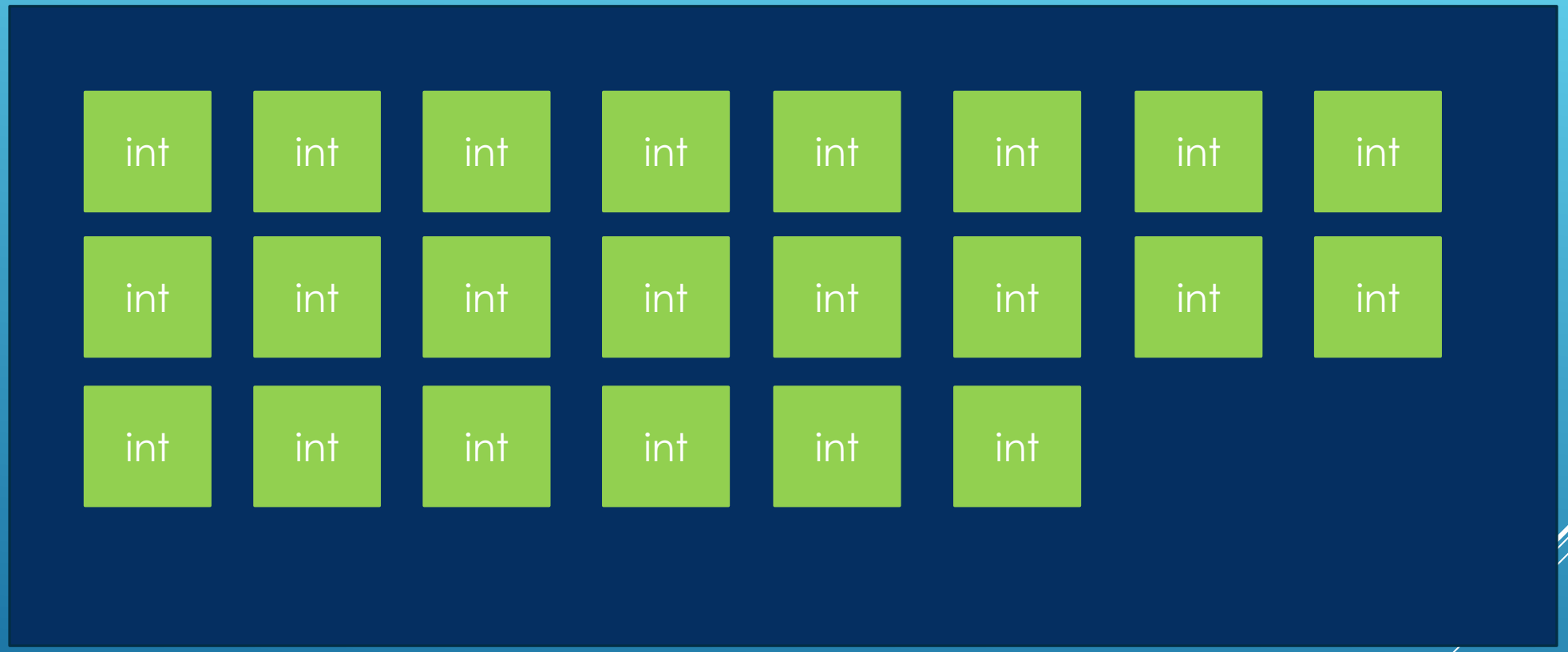
Replace entire switch with single array index access line

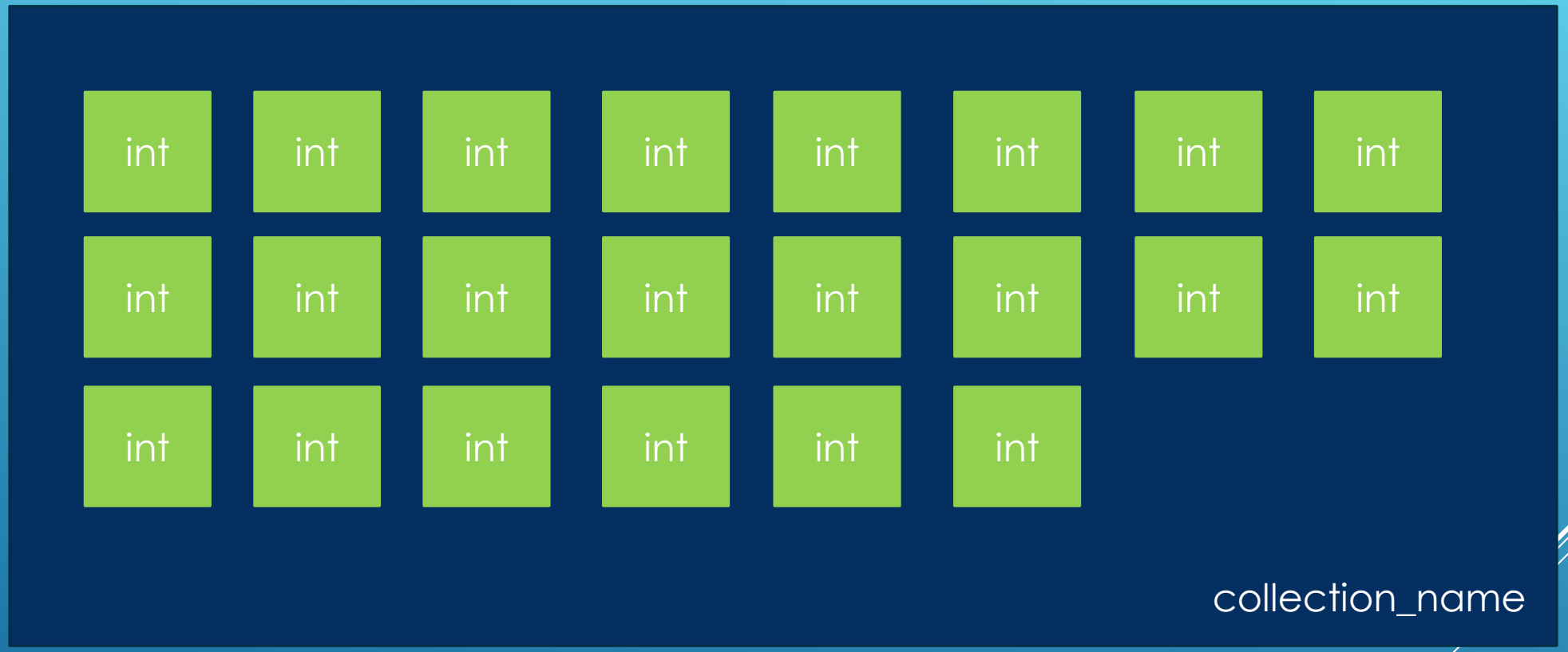
```
while(!end){
    std::cout << "Oh dear " <<name << ", I see ";

    size_t random = static_cast<size_t>((std::rand() % count) + starting_point);

    std::cout << predictions[random] << std::endl;


      /* switch (random(10)){ ...

    std::cout << "Do you want me to try again ? (Y | N) : ";

    char go_on;
    std::cin >> go_on;

    end = ((go_on == 'Y') || (go_on == 'y')) ? false : true;

}
```

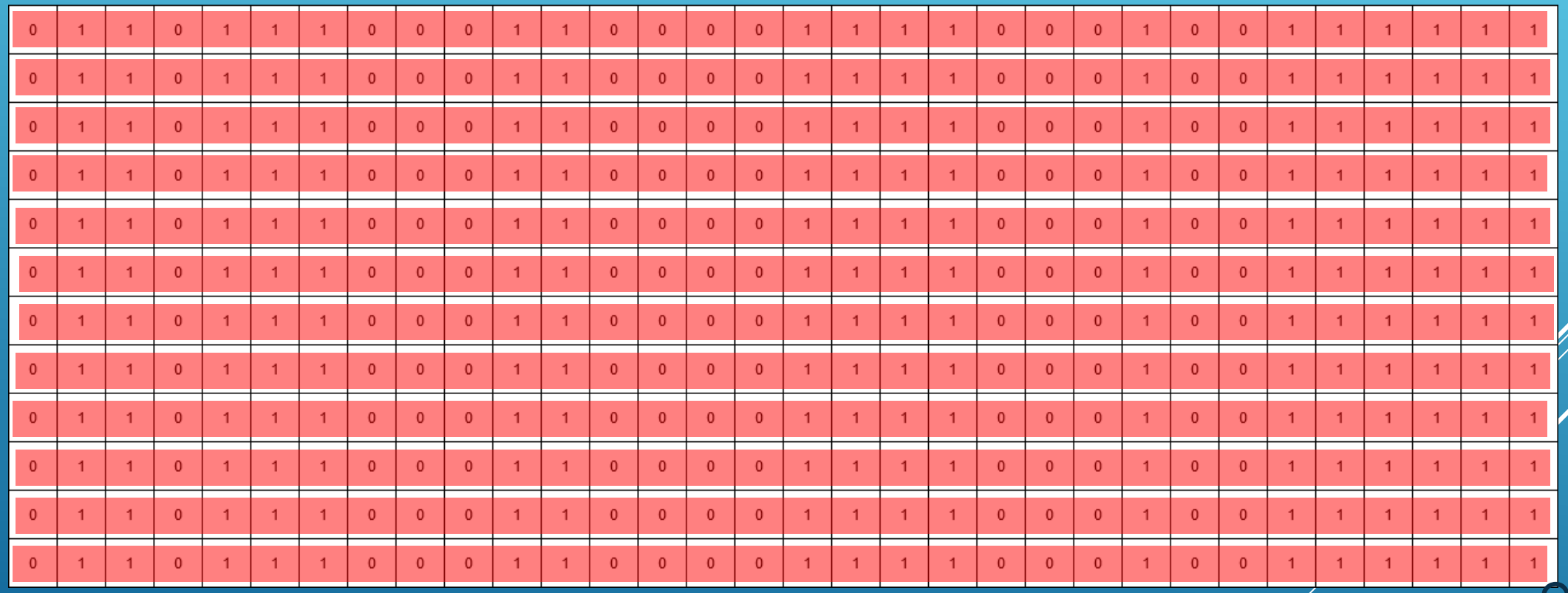
Slide intentionally left empty

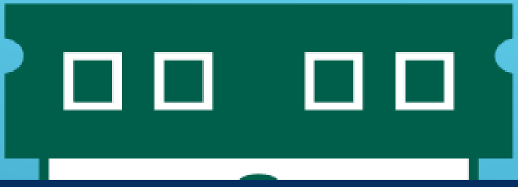
Arrays : Summary


int int int int int int int int
int int int int int int int int
int int int int int int

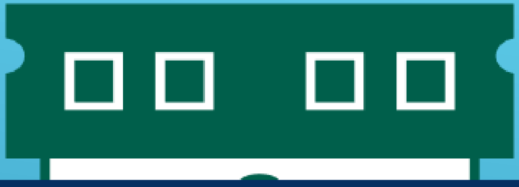

int
int
int
int
int
int
int
int
int
int
int
int
int
int
int
int
int
int
int
int
int
int
int
int


int
int
int
int
int
int
int
int
int
int
int
int
int
int
int
int
int
int
int
int
int
int
int
int
collection_name


0 1 1 0 1 1 1 0 0 0 1 1 0 0 0 0 1 1 1 1 0 0 0 1 0 0 1 1 1 1 1 1



0	1	1	0	1	1	1	0	0	0	1	1	0	0	0	0	1	1	1	1	0	0	0	1	0	0	1	1	1	1	1	1
0	1	1	0	1	1	1	0	0	0	1	1	0	0	0	0	1	1	1	1	0	0	0	1	0	0	1	1	1	1	1	1
0	1	1	0	1	1	1	0	0	0	1	1	0	0	0	0	1	1	1	1	0	0	0	1	0	0	1	1	1	1	1	1
0	1	1	0	1	1	1	0	0	0	1	1	0	0	0	0	1	1	1	1	0	0	0	1	0	0	1	1	1	1	1	1
0	1	1	0	1	1	1	0	0	0	1	1	0	0	0	0	1	1	1	1	0	0	0	1	0	0	1	1	1	1	1	1
0	1	1	0	1	1	1	0	0	0	1	1	0	0	0	0	1	1	1	1	0	0	0	1	0	0	1	1	1	1	1	1
0	1	1	0	1	1	1	0	0	0	1	1	0	0	0	0	1	1	1	1	0	0	0	1	0	0	1	1	1	1	1	1
0	1	1	0	1	1	1	0	0	0	1	1	0	0	0	0	1	1	1	1	0	0	0	1	0	0	1	1	1	1	1	1
0	1	1	0	1	1	1	0	0	0	1	1	0	0	0	0	1	1	1	1	0	0	0	1	0	0	1	1	1	1	1	1
0	1	1	0	1	1	1	0	0	0	1	1	0	0	0	0	1	1	1	1	0	0	0	1	0	0	1	1	1	1	1	1
0	1	1	0	1	1	1	0	0	0	1	1	0	0	0	0	1	1	1	1	0	0	0	1	0	0	1	1	1	1	1	1
0	1	1	0	1	1	1	0	0	0	1	1	0	0	0	0	1	1	1	1	0	0	0	1	0	0	1	1	1	1	1	1



0	1	1	0	1	1	1	0	0	0	1	1	0	0	0	0	1	1	1	1	0	0	0	1	0	0	1	1	1	1	1	1
0	1	1	0	1	1	1	0	0	0	1	1	0	0	0	0	1	1	1	1	0	0	0	1	0	0	1	1	1	1	1	1
0	1	1	0	1	1	1	0	0	0	1	1	0	0	0	0	1	1	1	1	0	0	0	1	0	0	1	1	1	1	1	1
0	1	1	0	1	1	1	0	0	0	1	1	0	0	0	0	1	1	1	1	0	0	0	1	0	0	1	1	1	1	1	1
0	1	1	0	1	1	1	0	0	0	1	1	0	0	0	0	1	1	1	1	0	0	0	1	0	0	1	1	1	1	1	1
0	1	1	0	1	1	1	0	0	0	1	1	0	0	0	0	1	1	1	1	0	0	0	1	0	0	1	1	1	1	1	1
0	1	1	0	1	1	1	0	0	0	1	1	0	0	0	0	1	1	1	1	0	0	0	1	0	0	1	1	1	1	1	1
0	1	1	0	1	1	1	0	0	0	1	1	0	0	0	0	1	1	1	1	0	0	0	1	0	0	1	1	1	1	1	1
0	1	1	0	1	1	1	0	0	0	1	1	0	0	0	0	1	1	1	1	0	0	0	1	0	0	1	1	1	1	1	1
0	1	1	0	1	1	1	0	0	0	1	1	0	0	0	0	1	1	1	1	0	0	0	1	0	0	1	1	1	1	1	1
0	1	1	0	1	1	1	0	0	0	1	1	0	0	0	0	1	1	1	1	0	0	0	1	0	0	1	1	1	1	1	1
0	1	1	0	1	1	1	0	0	0	1	1	0	0	0	0	1	1	1	1	0	0	0	1	0	0	1	1	1	1	1	1

collection_name

Declaring and Reading from an Array

```
//Declaring an array
int scores[10]; // An array storing 10 integers

//Reading values
std::cout << std::endl;
std::cout << "Reading out score values (manually) : " << std::endl;
std::cout << "scores[0] : " << scores[0] << std::endl;
std::cout << "scores[1] : " << scores[1] << std::endl;
//...
std::cout << "scores[9] : " << scores[9] << std::endl;
```

Initialize the array at declaration

```
//Declare and initialize at the same time
std::cout << std::endl;
std::cout << "Declare and initialize at the same time : " << std::endl;

double salaries[5] {12.7, 7.5, 13.2, 8.1, 9.3};

for(size_t i{0}; i < 5; ++i){
    std::cout << "salary[" << i << "] : " << salaries[i] << std::endl;
}
```

std::size() [C++17]

```
int scores[] {10,12,15,11,18,17,22,23,24};

//Can get the size with std::size
std::cout << "scores size : " << std::size(scores) << std::endl;

//Print data out
for( size_t i{0} ; i < std::size(scores) ; ++i){
    std::cout << "scores[" << i << "] : " << scores[i] << std::endl;
}
```

Character arrays

```
//Can also define a literal C string
std::cout << std::endl;
char message4 [] {"Hello"}; // An implicit '\0' character is appended to the
                            // end of the string, making it a c string
std::cout << "message4 : " << message4 << std::endl;
std::cout << "size : " << std::size(message4) << std::endl;


//Can even have spaces between characters
std::cout << std::endl;
char message5[] {"Hello World!" };
std::cout << "message5 : " << message5 << std::endl;
std::cout << "size : " << std::size(message5) << std::endl;
```

Multidimensional arrays

```
//You can omit the number of elements inside an [] when declaring a multi dimensional
// array but only one, and it has to be the left most.
// int packages2 [] [num_cols] will work, but int packages2 [] [] won't work and give a
//compile error

//const size_t num_rows{4};
const size_t num_cols{3};

int packages2 [] [num_cols] {
    {1,2,3},
    {4,5,6},
    {7,8,9},
    {10,11,12},
    {100,110,120} // Can add as many triplets in packages2 as we want
};

std::cout << std::endl;
std::cout << "Ommiting leftmost dimension for 2d array : " << std::endl;
for (size_t i{0} ; i < std::size(packages2) ; ++i){
    for( size_t j{0} ; j < std::size(packages2[i]) ; ++j){
        std::cout << "Item (" << i << "," << j << ") : " << packages2[i][j] << std::endl;
    }
}
```

2D array of characters

```
//Updating our prediction declaration in the fortune teller game
char predictions [] [90] {
    "a lot of kinds running in the backyard!",
    "a lot of empty beer bootles on your work table.",
    "you Partying too much with kids wearing weird clothes.",
    "you running away from something really scary",
    "clouds gathering in the sky and an army standing ready for war",
    "dogs running around in a deserted empty city",
    "a lot of cars stuck in a terrible traffic jam",
    "you sitting in the dark typing lots of lines of code on your dirty computer",
    "you yelling at your boss. And oh no! You get fired!",
    "you laughing your lungs out. I've never seen this before."

};
```

Slide intentionally left empty