



Diploma in

Computer Science



Arrays





GOAL

TARGET

IDEA

Objectives

Explain an array <

Identify where an array can be used <

Explore data manipulation using arrays <

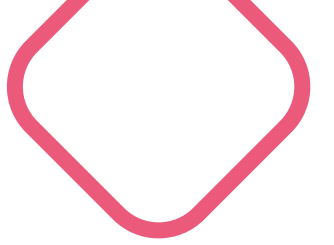
Apply array use guidelines in programming <



Lesson 3 Challenge Feedback

Create a program that demonstrates scope using the mayor and prime minister example from the lesson.

- Declare a global variable that represents the PM and a local variable for the mayor.
- The programme will display the text PM only when it cannot access the local variable.
- The programme will display the text mayor when it can access the local variable.



Arrays



**Why do we
need arrays?**



Why do we need arrays?



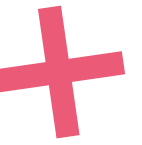
Array – a data structure consisting of a collection of elements

Example: A list of Fibonacci numbers

How would you store them?

Define an array that can hold 1000 elements.

Create a for loop that iterates through 1000 elements.





Arrays are used in linked lists, hash tables, search trees, queues, stacks, and strings.

With arrays, you can compile the indices at runtime.

Arrays are used for dynamic memory allocation within programs using emulation

In Maths, used to implement representation.
In databases, used to store records.



Why do we need arrays?



**DID YOU
KNOW?**

Processors are often
optimised for array
operations.



Another key use...

- Searching and sorting – arrays make these operations more efficient.
- This functionality is almost entirely dependent on arrays.
- Most search algorithms use arrays.



Array syntax



Uses of arrays

```
data_type arrayName[]={element1,  
element2, ...element n};
```

Example

```
data_type arrayName[n]={element1,  
element2, ...element n};
```

Example

```
data_type arrayName[n];
```

Example



Arrays are homogenous data structures – they store elements of the same type.



Accessing array data

How do we even store elements and keep track of the exact position in the array?

The *for loop* is best for counting.

Arrays are a contiguous block of memory, each with its own address.

The index refers to each element as a subunit of the variable.

Elements can be accessed individually but are part of the same data structure.



Accessing arrays example



In the set of marks for a class of 50, it is best if the marks are accessed one at a time rather than all poured out at once.



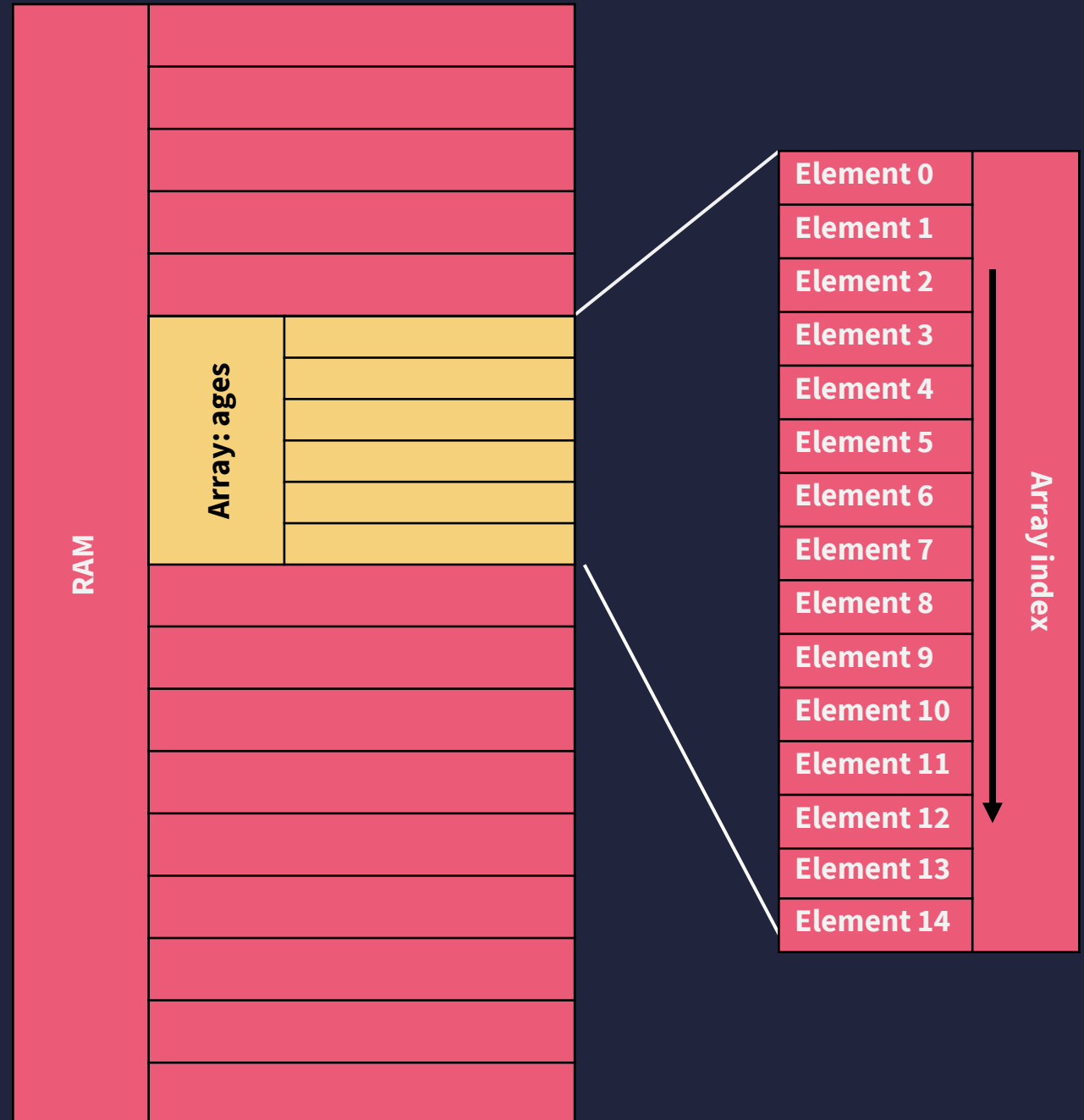


The array is a block of memory with a data type.

```
Int ages {14};
```

Example

From the C99 standard onwards, C supports variable length arrays.



Things to note...

- Once you declare the size and type of an array and compile the programme, you cannot change these attributes.
- Never attempt to access elements outside the bounds of the array.



Because arrays are stored in contiguous memory, they are extremely efficient at accessing values.





Array operations

- To assign a value to an array we use the index.
- Using an index means that you can randomly access any one of the elements.

```
ages[6] = 21;
```

Example



Performing regular arithmetic operations



You can use arrays to perform regular arithmetic operations.

Make sure the array index is always correct.



+ Performing regular arithmetic operations

```
#include<stdio.h>

int main()
{
    int Size, i, a[10], b[10];
    int Addition[10], Subtraction[10], Multiplication[10], Modulus[10];
    float Division[10];

    printf("\n Please Enter the Size of the Array\n");
    scanf("%d", &Size);

    printf("\nPlease Enter the First Array Elements\n");
    for(i = 0; i < Size; i++)
    {
        scanf("%d", &a[i]);
    }
}
```

Example



Code output

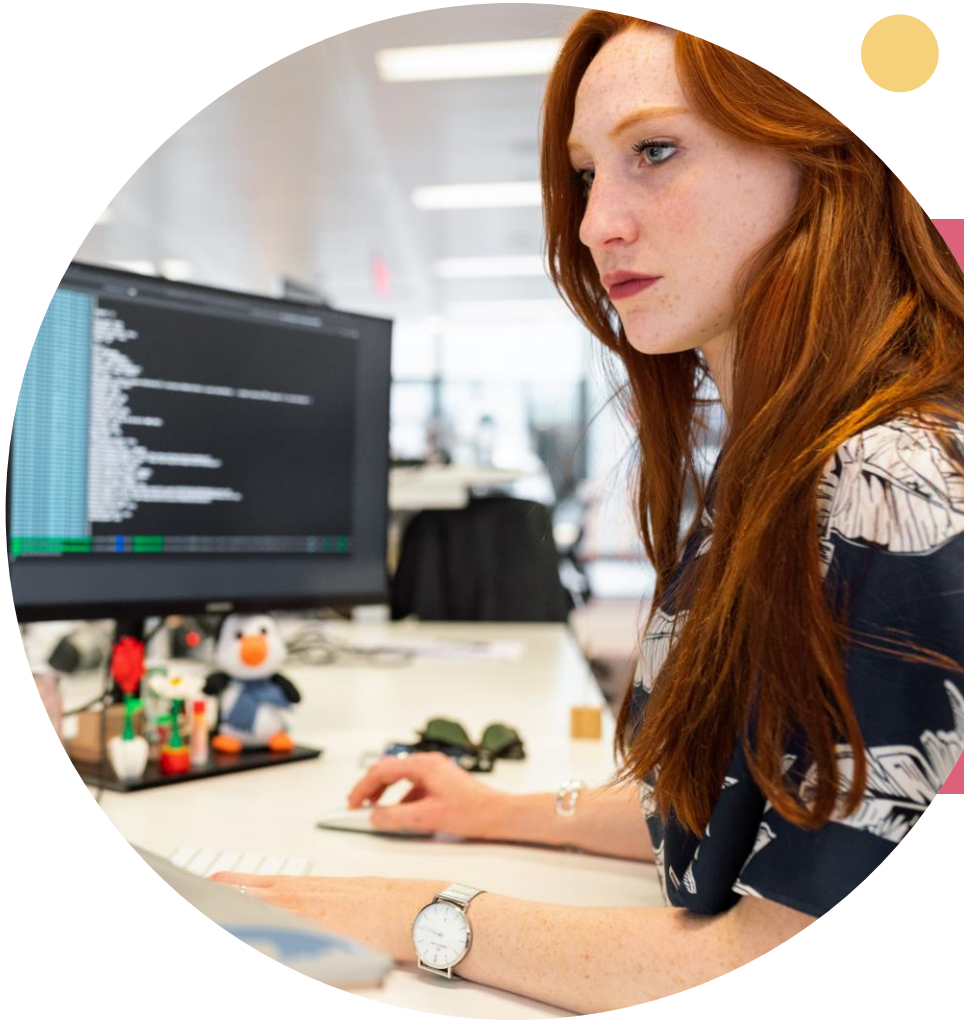
```
C:\Users\Samual Masuka\Desktop\demos\arrays.exe

Please Enter the Size of the Array
4

Please Enter the First Array Elements
1
2
3
4

Please Enter the Second Array Elements
4
3
2
1

Add      Sub      Multi     Div      Mod
5        -3        4         0.00     1
5        -1        6         0.00     2
5         1        6         1.00     1
5         3        4         4.00     0
-----
Process exited after 22.16 seconds with return value 0
Press any key to continue . . .
```



Logical operations

- Take note of the index number!
- Pay extra attention when declaring and initialising arrays to save yourself a lot of trouble.
- Keep your logic in check!



Character strings





Strings = arrays

In C, a string is a sequence of characters terminated with a null character which is a `\0`.

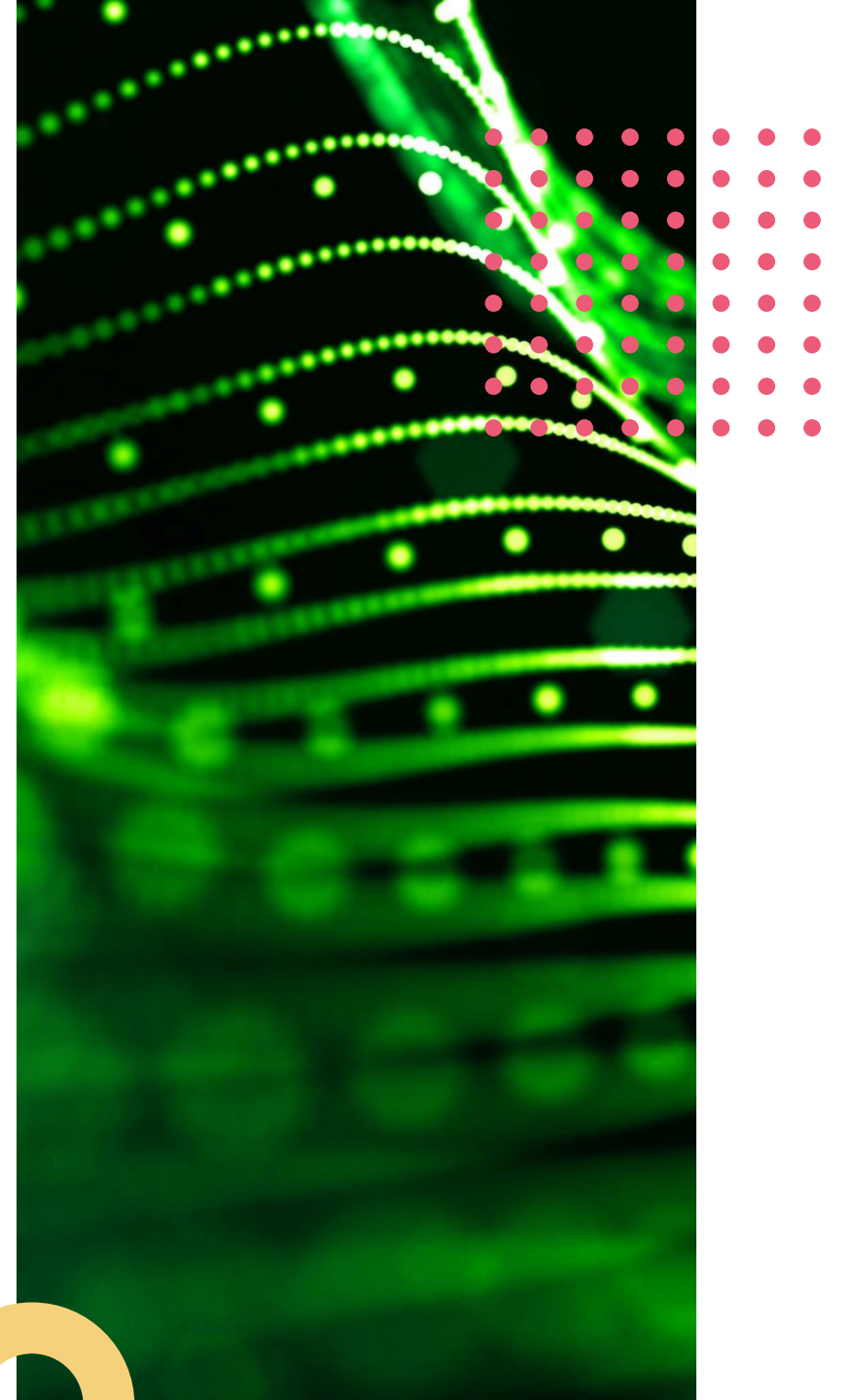
The way we define a string (square brackets) is the same as an array!



Strings = arrays

```
char names[5];
```

Example



Strings = arrays

```
char c[] = "abcd";
```

Example



Strings = arrays

```
char names[30] = "abcd";
```

Example



Strings = arrays

```
char names[] = {'a', 'b', 'c', 'd', '\0'};
```

Example



Strings = arrays

```
char names[5] = {'a', 'b', 'c', 'd', '\0'};
```

Example



Strings = arrays

```
char names[6] = "Samuel"; // won't work
```

Example

```
char names[6];  
str = "Samuel"; // won't work
```

Example





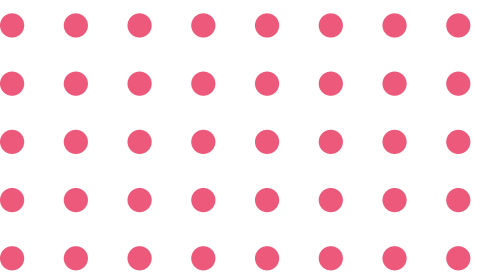
Jumping hurdles

- When reading strings, the program will read the sequence of characters that the user enters until it encounters white space.
- The way around this is a format specification known as the edit set conversion code, like this:

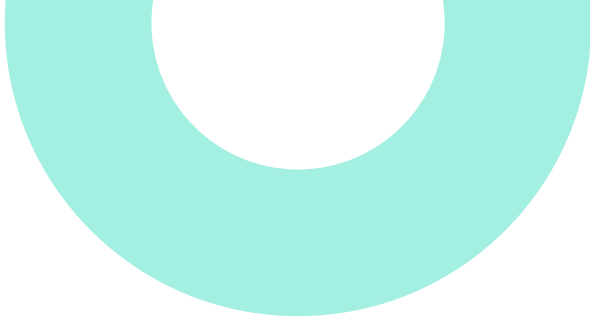
```
% [ . . . ]
```

Example





**There are so
many ways to
get around the
limitations of
programming
languages.**



The string.h header file



String operations

strcat	Concatenate two strings
strchr	String scanning operation
strcmp	Compare two strings
strcpy	Copy a string
strlen	Get string length
strncat	Concatenate one string with part of another
strncmp	Compare parts of two strings
strncpy	Copy part of a string
strrchr	String scanning operation



**Snap
& save**

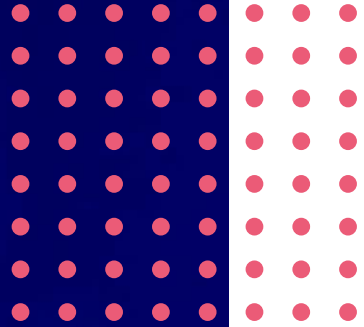




Strcat

```
char *strcat(char *destination, const char  
*source)
```

Example

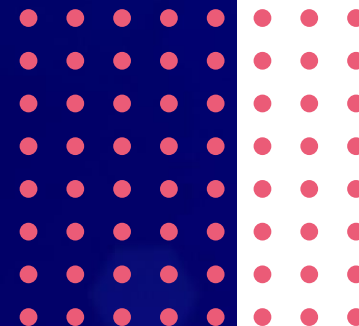




Strchr

```
char *strchr(const char *str, int c)
```

Example

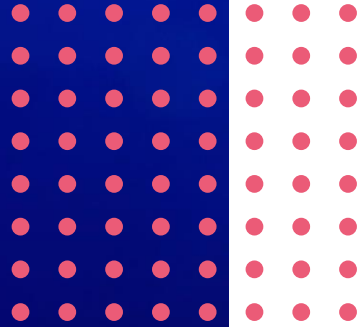




Strcmp

```
char *strcpy(char *restrict s1, const char  
*restrict s2);
```

Example

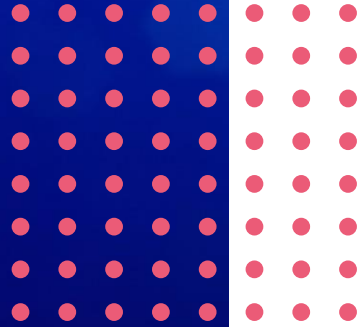




Strcpy

```
char *strcpy(char *restrict s1, const char  
*restrict s2);
```

Example

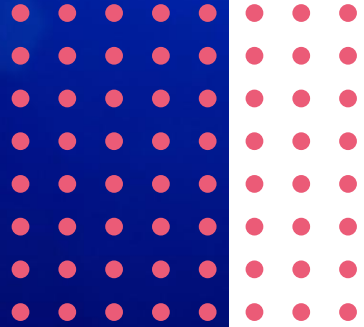




Strlen

```
size_t strlen(const char *s);
```

Example

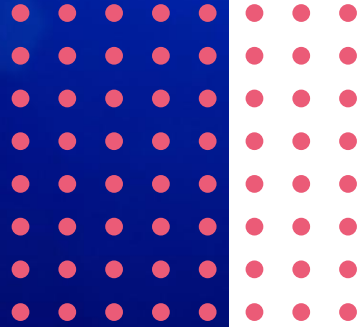




Strncat

```
char *strncat(char *restrict s1, const  
char *restrict s2, size_t n);
```

Example





Strncmp

```
int strncmp(const char *s1, const char *s2, size_t n);
```

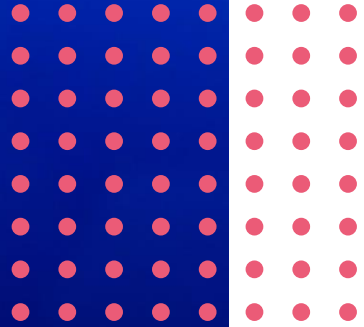
Example



Strncpy

```
char *strncpy(char *restrict s1, const  
char *restrict s2, size_t n);
```

Example

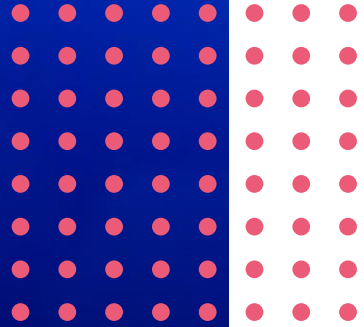




Strrchr

```
char *strrnchr(const char *s, int c);
```

Example





**Watch the length of
your strings – rather
have left-over unused
array indices than
string overflows!**



memchr	Find a byte in memory
memcmp	Compare bytes in memory
memcpy	Copy bytes in memory
memmove	Copy bytes in memory with overlapping areas
memset	Set bytes in memory
strcoll	Compare bytes according to a locale-specific collating sequence
strcspn	Get the length of a complementary substring
strerror	Get error message
strpbrk	Scan a string for a byte
strspn	Get the length of a substring
strstr	Find a substring
strtok	Split a string into tokens
strxfrm	Transform string





Array dimensions



Single
dimension
arrays – one
long chain



**Multi dimension
arrays – an
array of arrays
in tabular form**





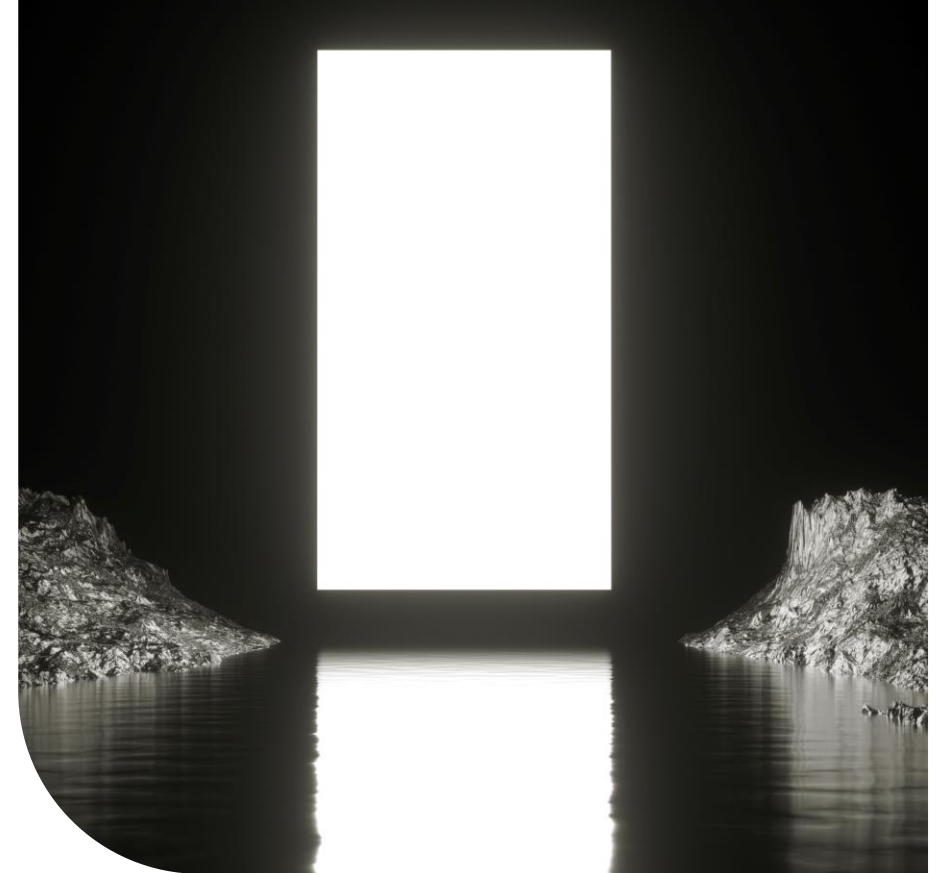
Multi dimension arrays

```
data_type array_name[x][y];
```

Example

```
int twoD_array[10][20];
```

Example





Multi dimension arrays

```
data_type array_name[x][y][z];
```

Example



The wonder of multi dimension arrays...



On a screen: can use an array to display 8 bits of colour in each pixel

In graphs: can use an array to create three-dimensional graphs