

[the academy\_of\_code]

## Useful links for algorithms and data structures

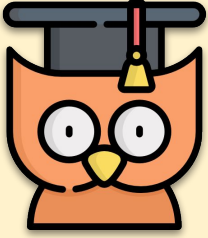
- [https://cdn.preterhuman.net/texts/math/Data\\_Structure\\_And\\_Algorithms/Teach%20Yourself%20Data%20Structures%20And%20Algorithms%20In%2024%20hours%20-%20Robert%20Lafore.pdf](https://cdn.preterhuman.net/texts/math/Data_Structure_And_Algorithms/Teach%20Yourself%20Data%20Structures%20And%20Algorithms%20In%2024%20hours%20-%20Robert%20Lafore.pdf)
  - <http://algorithms.us/data-structures-basics/>

**[the academy\_of\_code]**

**Grade ?**

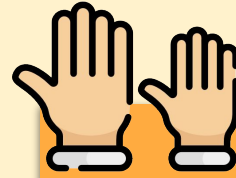
**Unit Problem Solving**

# Lesson 1 - Introduction to functions



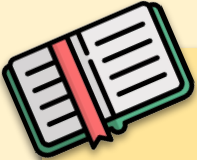
## Learning Outcomes:

- How to solve common programming problems



## REMEMBER:

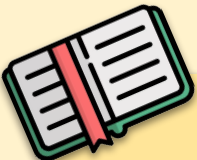
Put up your hand. We love to help!



## Welcome Back!

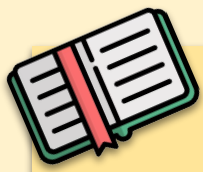
In this unit we are going to move away from the graphical programming we have been doing up to now and focus just on solving computational programming problems.

To do this we are going to start by going back to the basics and just look at `void setup()` and we are going to ignore `void draw()`.



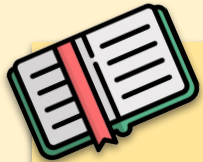
## What we need to know

- First we need to remember that `void setup()` runs only once. This means that once the code has finished nothing more is going to be done.
- Next that `void draw()` runs continuously and will keep running the code until the program is exited.
- These two facts mean that it is much easier to control the flow of our program if we don't use `void draw()`.
- Because of this and the fact that `void draw()` is an addition made by Processing that isn't usually in a programming language, we are going to stop using `void draw()` to solve the problems we will be facing in this unit.



# Functions

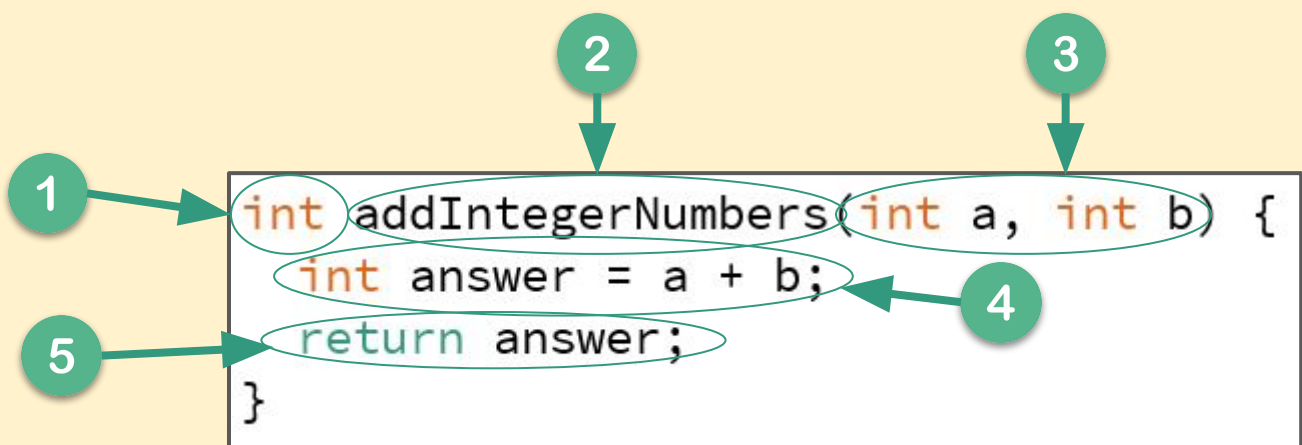
Functions are an important part of coding. Functions have several different purposes. The first is that it is an easy way to break our code into smaller chunks that are easier to understand and to code. The second is that it allows us to reuse chunks of code, thus reducing how much code we need to write and how much code we need to maintain.



## How do functions work

Below is an example of a function that adds two integer numbers. We are going to look at each part of this function.

- 1 This is the return type of the function. In this example the return type is an int. Other return types are boolean, float, String. Up to now many of the functions we wrote didn't return anything in this case we use void.
- 2 This is the name of the function. It is important that this name is clear and makes sense. Function names are written in lower camel case, meaning the first word starts with a lowercase letter and any other words begin with an upper case letter.
- 3 This is the parameter list. This is the information that we pass into the function. This section can contain as many parameters as we want, but it is a good idea to simplify it and reduce the number of parameters used. This can also be left blank
- 4 In the main part of the function we solve the problem the function is intending to solve. This may simply be one line of code or many depending on the problem.
- 5 Finally, if our return type is not void we need to return our answer.





## Let's write some functions

- 1 Write a function that takes two integers as parameters. Subtract the second number from the first number and return the answer in the form of an integer. Our `void setup()` code should look something like this.

```
void setup() {  
    int answer = subtractIntegers(8, 6);  
    println(answer);  
}
```

- 2 Next we are going to write a function that takes a single float and returns a boolean. If the float is less than 5 it returns true, otherwise return false.
- 3 Write a function that takes in a single int and returns a string. This string should be the day of the week to number corresponds to, i.e. 1 = "Monday", 2 = "Tuesday", etc. If the number does not correspond to a day of the week return some sort of error message.
- 4 Write a function that takes a string and doesn't return anything. (Hint: void). The function should print the string to the console using `println()`.
- 5 Write a function to determine if a float is a whole number. This function should take a float as a parameter and return a boolean. You will need to use the modulo operator (%) for this. The modulo operator divides one number by another and gives you the remainder.

The if statement will look something like this: `if (number % 10 == 0)`

This will equal true for numbers like 10,20,30 and will equal false for 4, 7, 18, 29.

**Spoilers:** The code for these

functions are on the next page if you get stuck.



**REMEMBER:**

**Show your work to a tutor  
when you are finished**



## Spoilers

1

```
int subtractIntegers(int a, int b) {  
    int answer = a - b;  
    return answer;  
}
```

2

```
boolean isLessThanFive(float a) {  
    if (a < 5) {  
        return true;  
    } else {  
        return false;  
    }  
}
```

3

```
String dayOfTheWeek (int day) {  
    if (day == 1) {  
        return "Monday";  
    } else if (day == 2) {  
        return "Tuesday";  
    } else if (day == 3) {  
        return "Wednesday";  
    } else if (day == 4) {  
        return "Thursday";  
    } else if (day == 5) {  
        return "Friday";  
    } else if (day == 6) {  
        return "Saturday";  
    } else if (day == 7) {  
        return "Sunday";  
    } else {  
        return "Not a day of the week";  
    }  
}
```



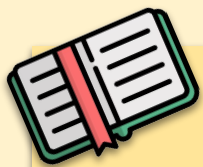
## Spoilers

4

```
void printToConsole(String text) {  
    println(text);  
}
```

5

```
boolean isWholeNumber (float a) {  
    if (a % 1 == 0) {  
        return true;  
    }  
    return false;  
}
```



## Program flow

An important concept in programming is the flow of the program. In simple terms this is just what happens when and in what order. For example, when we first start our programs our `void setup()` code is run. Next our `void draw()` code is run repeatedly. Within this we use if statements and for loops to affect how our code is run.

Using functions, we have come across a new command that can affect the flow of our program. That is the **return** statement. The return statement can be used in functions and will exit the function and return to the main program. Because of this we can write two pieces of code that might look different, but in fact work the exact same way.

1

```
boolean isWholeNumber (float a) {  
    if (a % 1 == 0) {  
        return true;  
    }  
    return false;  
}
```

2

```
boolean isWholeNumber (float a) {  
    if (a % 1 == 0) {  
        return true;  
    } else {  
        return false;  
    }  
}
```

The two pieces of code above will work in the same way. In my opinion the first piece of code is more straightforward and would be my preference. This becomes more important and obvious in larger pieces of code. The most important thing is to think through exactly how your code will run and how it is supposed to run.





## Sample Problems

Now that we have gone through the basics needed to write a function and written a few ourselves, it is time to solve many more problems.

The link is: <https://edabit.com/collection/GsvaovowMqDRPWmK3>

This is a collection of problems that have been selected as a good introduction to problem solving. You can **either** *sign up and solve these on edabit* **or** *write the code in Processing and show the tutors*.

If you are finding the questions too easy talk to your tutor.

- 1 Write a function that converts minutes into seconds. It should take an integer *minutes* as a parameter and return its equivalent in seconds.
- 2 Write a function that converts hours into seconds. It should take an integer *hours* as a parameter and return its equivalent in seconds.
- 3 Write a function that returns the sum of two numbers. It should take two integers as arguments and return the two numbers added together.
- 4 Write a function that returns the next number from the integer passed. The function should take a number as an argument, increment it by 1 and return the result.
- 5 Write a function that calculates the perimeter of a rectangle. It should take two integers, *height* and *width*, as parameters and return the perimeter.

Write a function that finds the maximum length of a triangle's third edge. The function should take 2 integers as arguments. **Hint:** Maximum length of a third edge = side1 + side2 -1



## Solving problems

Write a function that takes two integers as arguments and checks if they are equal to each other. If they are equal, the function should return true, otherwise return false. **Hint:** The return value of this function is a boolean.

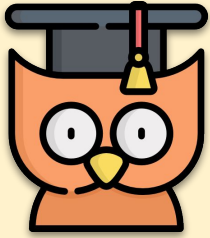
Write a function that checks if a number is less than or equal to zero. It should take an integer as an argument and return true if it is less than or equal to zero. Otherwise return false.

Write a function that checks if the sum of two numbers is less than 100. The function should take 2 integers as arguments, and return true if their sum is less than 100. Otherwise return false.

Write a function that returns the remainder from two numbers. It should take two integers as parameters. The first parameter divided by the second parameter will have a remainder. Return this number. **Hint:** Use the modulo % operator.

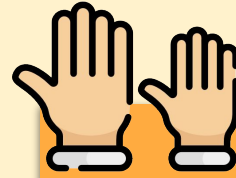
Write a function that takes a name (string) as a parameter and returns a greeting in the form of a string. For example: `helloName("Niall")` → "Hello Niall!"

# Lesson 2 - Using Arrays




## Learning Outcomes:

- How to solve common programming problems using arrays



## REMEMBER:

Put up your hand. We love to help!



## Arrays

By now you should have learnt about how to use arrays. In this section we are going to do a revision of arrays and explore the sorts of problems that can be solved using arrays.



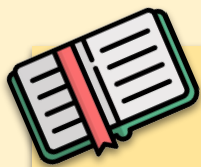
## Arrays - Declaring and initialising

- Arrays are used to store a collection of data of the same variable type. E.g. **int**, **float**, **boolean**, **String**, etc.
- Like other variable types we need to declare and initialise our arrays.
- The two options to declare and initialise in the same line are below. The first creates an array of size 3 and doesn't set the value. The second creates an array of size 3, but specifies the values of each item.

```
int[] numbersOne = new int[3];  
int[] numbersTwo = {0,0,0};
```

- If we declare and initialise on separate lines it will look as below.

```
int[] numbersOne;  
numbersOne = new int[3];  
int[] numbersTwo;  
numbersTwo = new int[]{0, 0, 0};
```



## Array - Accessing

To access data items in the array we use square brackets [ ] and we put a number to indicate the position of the item we want. The line below shows us accessing the 0th element in the array using the println function to output the result the console.

```
println(numbersOne[0]);
```



### Expert Tip

The element of an array start at 0.

## Array - Changing

To change the value of a specific element we must first access the element as above and then set it to a new value.

```
numbersOne[0] = 5;
```

## Array - Length

Often we need to know the length of the array to solve problems. This is simple to get. The line below will print out the length of our array.

```
println(numbersOne.length);
```



### Quick tasks to practice

1

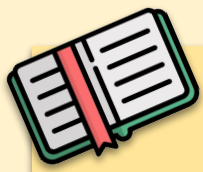
Declare and initialise two different arrays. Make them different types and use different methods to create them.

2

Print out three values from the array. **Challenge:** Use random to print out a random element from the array. Make sure that this won't cause errors.

3

Change two values in the array and print out the length of the two arrays.



## Looping through an array

Many problems that use arrays will require that you loop through the whole array. The code below will loop through the whole array, beginning at zero.

```
for (int i = 0; i < numbersOne.length; i++) {  
    println(numbersOne[i]);  
}
```

The other common loop you might need is looping through the array backwards. I.e. starting at the largest element. The code below will do that.

```
for (int i = numbersOne.length - 1; i >= 0; i--) {  
    println(numbersOne[i]);  
}
```



### Expert Tip

Learn this code or save it somewhere you are easily able to reference it. You will use it a lot.

## Errors

The most common error you are likely to see when using arrays is the array index out of bounds exception. This occurs when you try and access an element that is not in the array.

For example if I have an array of length four and write the code below.

```
println(numbersTwo[4]);
```

I will get the error below, because remember the numbers in an array start from 0.

```
ArrayIndexOutOfBoundsException: 4
```

Usually this problem is caused by an off by one error. Like in the second for loop forgetting the -1 will cause this problem or writing > where you should write >=.



## Let's answer some questions

- 1 Declare and initialise two arrays. Print out the values of the array starting at element 0. Print out the values of the array starting at the last element.
- 2 Create a situation in which you get the `ArrayIndexOutOfBoundsException`.

Try solving the five edabit questions below

Squares and Cubes - <https://edabit.com/challenge/qaKffo5AZo2RMcKyN>

Use `Math.sqrt()` and `Math.cbrt()` to find the squares and cubes

Check if an Array Contains a Given Number -

<https://edabit.com/challenge/hAtARtyLzAHb2TEP5>

You will need to loop through the array

Shapes With N Sides - <https://edabit.com/challenge/odJPfYRD3kSpE45Jf>

This question can be done without using arrays. You can do it that way first and then try solving it using arrays.

Index Multiplier - <https://edabit.com/challenge/YMLm3DpuZXHFDAnou>

Use a loop and some maths

Multiply by Length - <https://edabit.com/challenge/Mr48aLdKMxpGrdTXK>

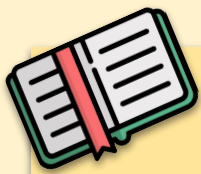
Use a loop and some maths

Find some more array questions to practice.



**REMEMBER:**

**Show your work to a tutor  
when you are finished**



## Copying an array

We are going to look at three correct methods for copying an array and one generally incorrect method.

The **first method** requires that you create a second array, the same length as the array you want to copy. Then you need to loop through the array and copy each individual value. The code for this is below.

```
int[] c = {1, 2, 3};  
int[] d = new int[c.length];  
for (int i = 0; i < d.length; i++) {  
    d[i] = c[i];  
}
```

The **second method** also requires that you create a second array, the same length as the array you want to copy. You will use the **System.arraycopy** function to copy the array.

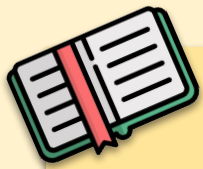
**System.arraycopy** takes 5 parameters. The first parameter is the array you want to copy, the second is the position to start copying from (when copying it all, this will always be 0). The third parameter is the array you want to copy it to. The fourth parameter is the position you want start copying the element into (in a new array this will be 0). The fifth parameter is how many elements you want to copy (to copy the whole array this will always be arrayName.length)

```
int[] g = {1, 2, 3};  
int[] h = new int[g.length];  
System.arraycopy(g, 0, h, 0, h.length);
```

The **third method** is the simplest. You will use the clone function of the array. This will clone the array into the new array.

```
int[] e = {1, 2, 3};  
int[] f = e.clone();
```





## Which method to use

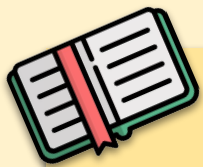
We looked at three different methods to get the same result, but why would we use one over any of the others.

The third method is obviously the simplest to write, but it also gives us the least flexibility. We can only create an exact copy of the original array.

The second method gives more flexibility. We can specify which section of the array we want to copy and where we want to copy it to, but it will take slightly longer to write.

The first method offers the most flexibility, but is also the longest to write. This method gives us total control over what gets copied.

The means that you need to be familiar with all three methods and know when and how to use them.

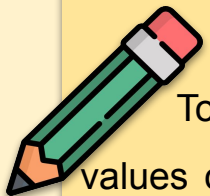


## How not to copy an array

The last method we are going to look at has some potentially problematic side effects. The method is simple and similar to how you would copy an **int** or **float**. The code is below.

```
int[] a = {1, 2, 3};  
int[] b = a;
```

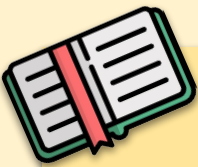
The problem with this method is that it doesn't actually create a new array with a new set of values. All it does is creates a new name (b) to look at the same array values.



To see this effect, create the two arrays using the code above. Print out all the values of the arrays. Change a single value in one of the arrays and print out both arrays again.

The in depth reason for this will be explored in later lessons.



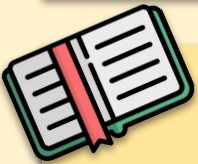
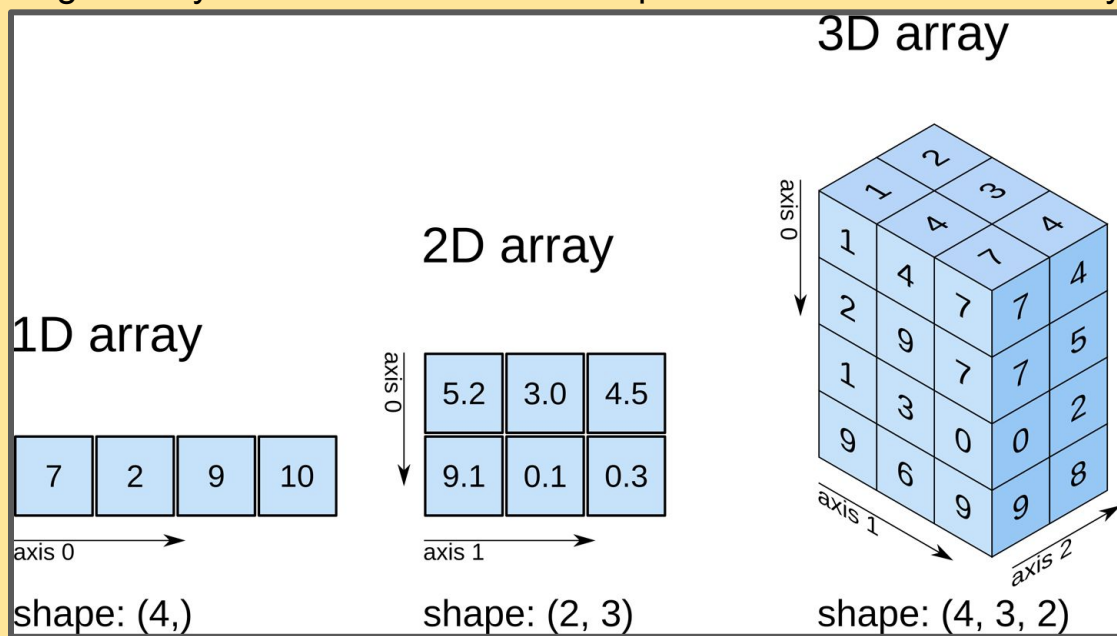


## Multidimensional Arrays

The arrays we have been looking at so far are 1-dimensional arrays. This means that they are just like a list. We can see this in the 1D array image below. If we want to access the number 9, the code would be **arrayName[2]**.

Next we see the 2-dimensional array. This is an array of arrays. If we write **arrayName[0]** for this array we will get the whole top line, {5.2, 3.0, 4.5}. If we want to access the number 0.3, we would write **arrayName[1][2]**.

The third array is a 3-dimensional array. At this point it can begin to get a little bit more difficult to picture. To access the 3, we would write **arrayName[2][1][0]**. We can continue to make larger dimensions of arrays, but it gets progressively more difficult to follow and in general you are able to solve most problems with 1D or 2D arrays.



## Additional Study

In this lesson we have covered the basics of arrays and how to use them. As part of this we also used for loops and briefly discussed multidimensional arrays. There is a lot more to learn about both arrays, multidimensional arrays and for loops and how to use them to solve programming problems that we did not cover in this lesson. We will discuss these uses in future lessons as we cover more programming topics.



## Let's write some code

- 1 Declare and initialise two arrays. Print out the values of the array starting at element 0. Print out the values of the array starting at the last element.
- 2 Create a situation in which you get the `ArrayIndexOutOfBoundsException`.
- 3 Use at least two different methods to copy an array into a new array.

4

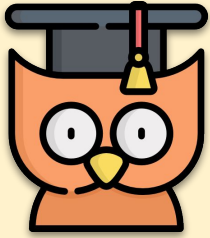
5



**REMEMBER:**

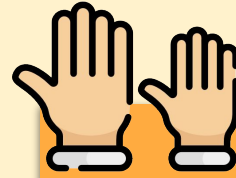
**Show your work to a tutor  
when you are finished**

# Lesson 2 - Strings



## Learning Outcomes:

- How to solve common programming problems using strings



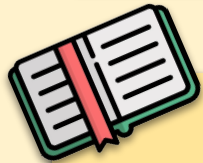
## REMEMBER:

**Put up your hand. We love to help!**



## Strings

You have already used strings in previous lessons to put text on the screen. This is a basic use of strings, but there is a lot more you can do with strings. In this section we are going to look at what you can do with strings, how to do it and how to solve questions using strings.



## Strings

Strings are easy to declare and initialise. They are very to ints or floats. The one important consideration is that you use double quotes (“”)

```
String str1 = "hello";
```

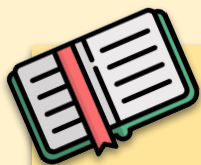
If we have a string we often want to print out the result. In the Processing this is easy, we can use either the print or the println function. The println function is the same as print, except that it adds a new line at the end.

```
print(str1);  
println(str1);
```

If you want to print out a string in Java, but not in Processing, it is a little more difficult. You need to change the code to look like below. This code will also work in Processing.

```
System.out.print(str1);  
System.out.println(str1);
```

**Task: Try to use each of the functions above**



## Strings functions

Much of the difficulty in using strings is being able to correctly use and remember all of the functions that are available to you. In this section we are going to look at and briefly explain how some of these functions work. Try using these functions first in Processing, changing them around and then try doing the challenges that are linked.

- There are two different methods for concatenating or putting strings together. The first uses the `+` symbol to add the strings together. The second uses the **concat** function to concatenate the two strings. This can also be done without the variables. **Challenge:** <https://edabit.com/challenge/JAtN6KLtahAkmT3n3>

```
String str1 = "hello";
String str2 = "student";
print(str1+str2);
// prints helloworld
```

```
String str1 = "hello";
String str2 = "student";
print(str1.concat(str2));
// prints helloworld
```

- We can use the **isEmpty** function to check if the string is empty. This will return a boolean, either true or false. **Challenge:** <https://edabit.com/challenge/wr8zTBNNelTspmLLT>

```
String str1 = "Hello";
print(str1.isEmpty());
// prints false
```

- To find the length of a string we use the **length** function. This returns the integer length of the string. **Hint:** How is this different to an array?

**Challenge:** <https://edabit.com/challenge/a9Shdt64Ak2Hwq7oP>

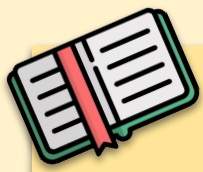
```
String str1 = "hello";
print(str1.length());
// prints 5
```

- The functions **endsWith** and **startsWith** both return a boolean. This will tell you if the string starts or ends with a certain string of characters.

**Challenge:** <https://edabit.com/challenge/pzLMEsMpbCLsPXqy2>

```
String str1 = "Hello";
print(str1.startsWith("lo"));
// prints false
```

```
String str1 = "Hello";
print(str1.endsWith("lo"));
// prints true
```



## Strings functions

- If we need to compare two strings there are a couple of ways of doing this. This most straightforward way is to use the **equals** function. This returns a boolean. **N.B.** If one string has a capital and the other doesn't, they are not equal.

```
String str1 = "hello";  
print(str1.equals("hello"));  
// prints true
```

- If we need to change the case of our string we can use either the **toUpperCase** or **toLowerCase** function. **Challenge:** <https://edabit.com/challenge/D6Lcut2s2gEzdCPvv>

```
String str1 = "Hello";  
print(str1.toLowerCase());  
// prints hello
```

```
String str1 = "Hello";  
print(str1.toUpperCase());  
// prints HELLO
```

- We can use the **contains** function to see if a string contains a specific substring. This also returns a boolean. **Challenge:** <https://edabit.com/challenge/YG5hYruH8TfqeZ3dc>

```
String str1 = "Hello";  
print(str1.contains("lo"));  
// prints true
```

- **Challenge:** <https://edabit.com/challenge/n22eru9bm8LMcBdYt>
- Hint look at arrays and concatenating