Coding

3 parts

1.C# coding
2. Xaml coding
3. Combine both

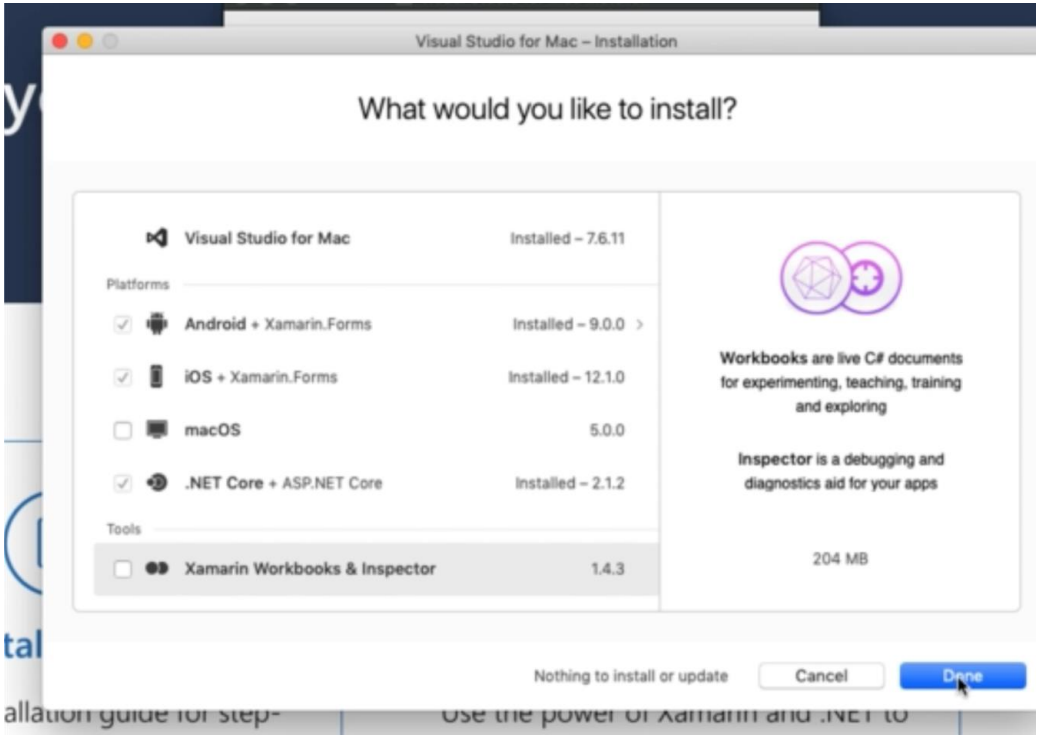
Installation Guide for Mac

Select 3 things

-Android + Xamarin.forms

-iOS + Xamarin.forms

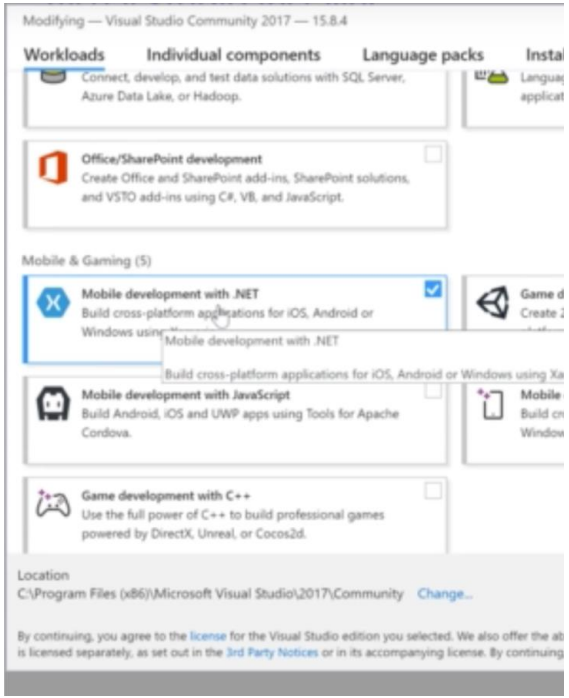
-.NET Core + ASP.NET Core



Installation guide for Windows

Click on Mobile Development with .NET

Launch

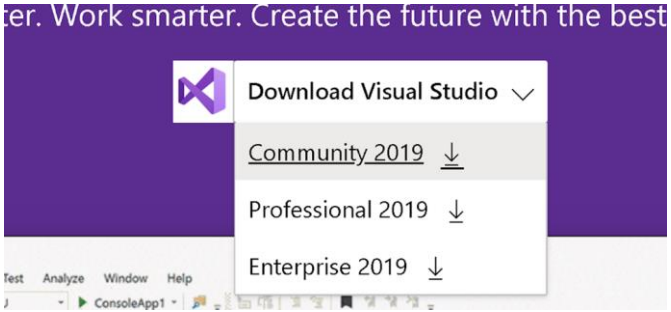


Installation Guide

Google search for ‘Visual Studio Code 2019’

Click on the link that goes to microsoft.com

Click on the community version 2019 as shown below



What is Required

Windows Computer (limited to Android app)

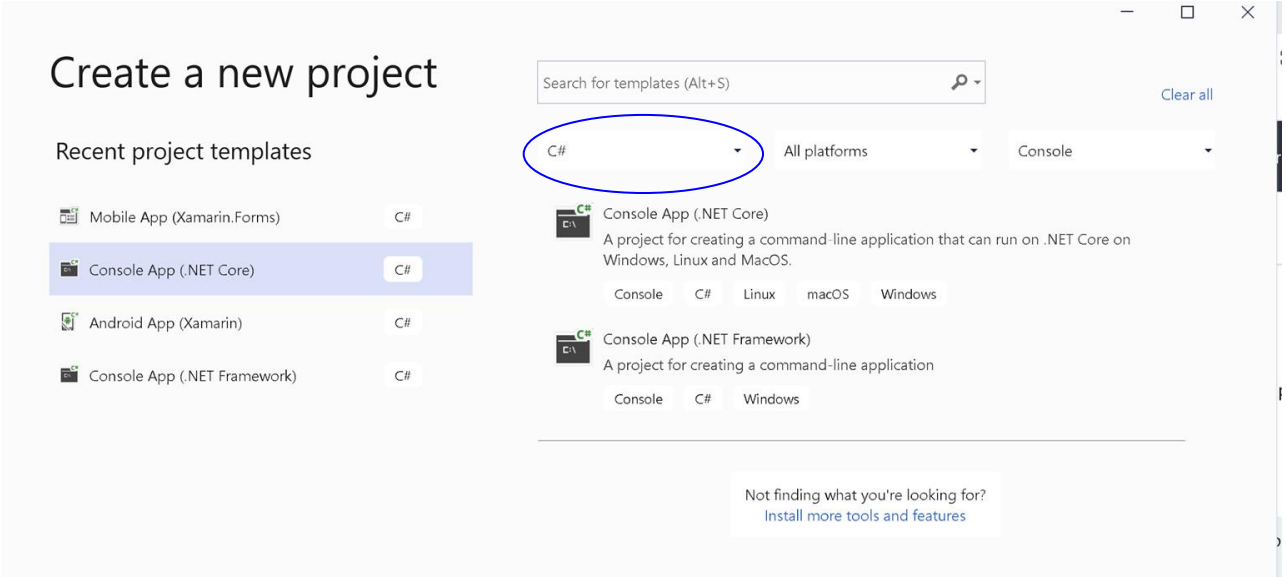
Mac Computer (if you want to do IOS app)

Visual studio code 2019

Basics of coding
C#

Setting up Console

1. Open visual studios 2019
2. Click on create new project
3. Ensure the language is C# (see the blue circle in the diagram below)

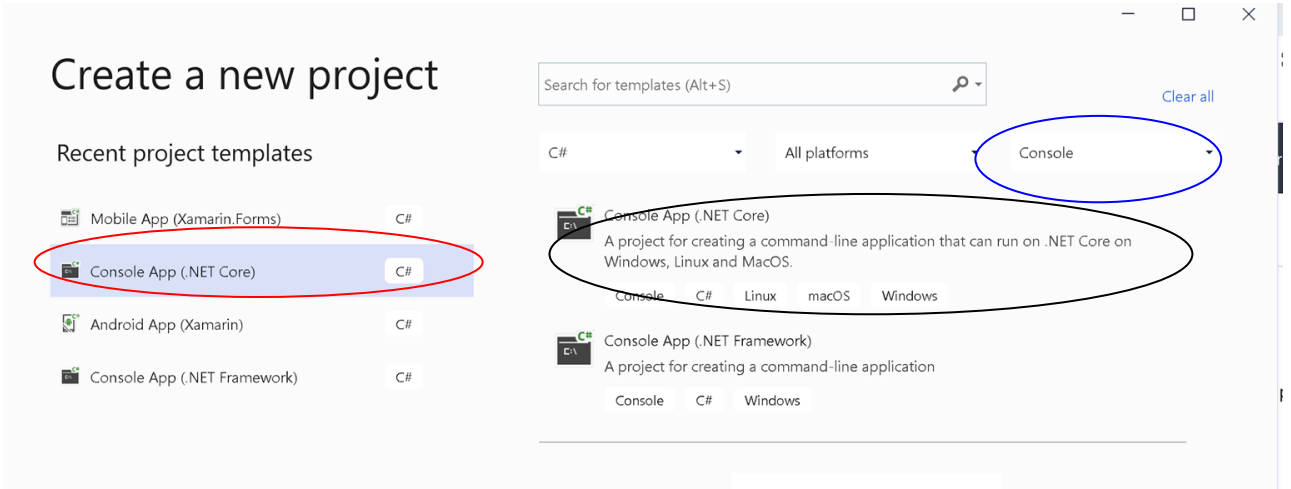


Setting up Console

4. Choose console for the 3rd right option (ref to blue circle)

5. Choose console app (see red circle)

6. On the right choose Console App .(net core) for windows macOS (see black circle)



Hello World

1. Below the Main method type

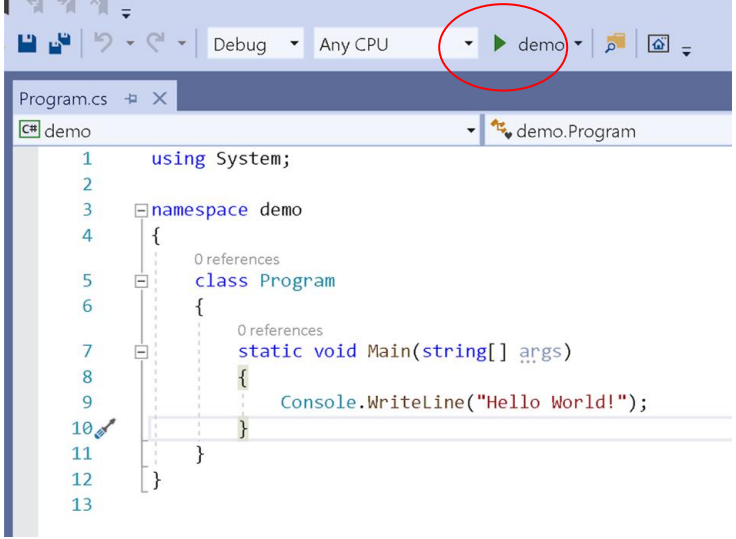
Console.WriteLine(“Hello World!”);

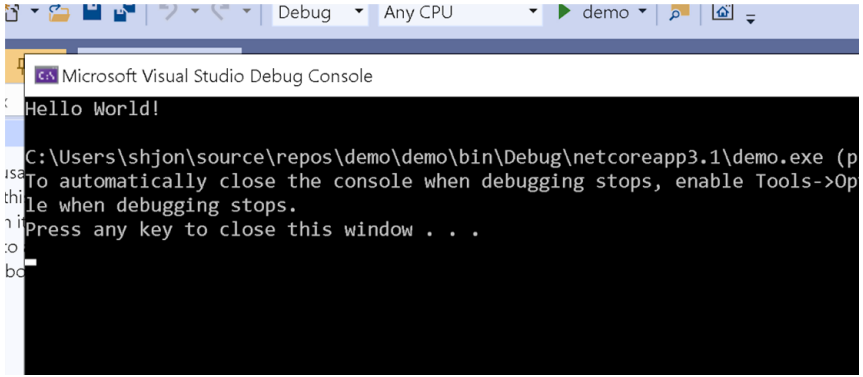
1. Click at the green play button at the top

(see red circle) this is the console.

1. The console/cmd prompt is displayed.

Hello world is displayed.





What do you need to know

- Data Types (Primitive Data Types vs Non Primitive Data Types)
- Operators
- Taking user input
- Control Flow statements(If/else statements, for/while loops, exception handling)
- Object Oriented Programming(OOP)

What is a Variable or a constant?

Variable is a name that we give to a storage location in memory

Constant is a immutable value which cannot be changed throughout the application

Declaring Variables and Constants

To declare a *var*

Format: Type Identifier (= value);

Eg1 int num;

Eg2 int Num2 = 2;

In eg 1, we declare an integer which is a type and an identifier (basically a name)

And semi colon to end the expression. Variable currently has no value (null)

In eg 2, we declare an type integer an an identifier called num2 to a value 2. So Num2 has a value of 2. Note that it is case sensitive so num2 and Num2 are 2 different variables.

Declaring a constant

To declare a *const*

Format: Const Type Identifier (=value);

Eg 1

const float pi = 3.14;

So in this example, we declare the keyword const which is constant, then we declare the type which in this case is float (will be covered later what exactly is float) and a identifier which is a name ‘pi’ and we assign pi to the value 3.14 and finally semi colon.

Identifiers

- Cannot start with a number eg 1num
- No whitespace eg Num 1
- No reserved words eg int (refer to pic) means these words are special Words that are used for fns etc.
- Case sensitive e.g ABC and abc are different
- Only alphanumeric characters allowed(A-Z,0-9)

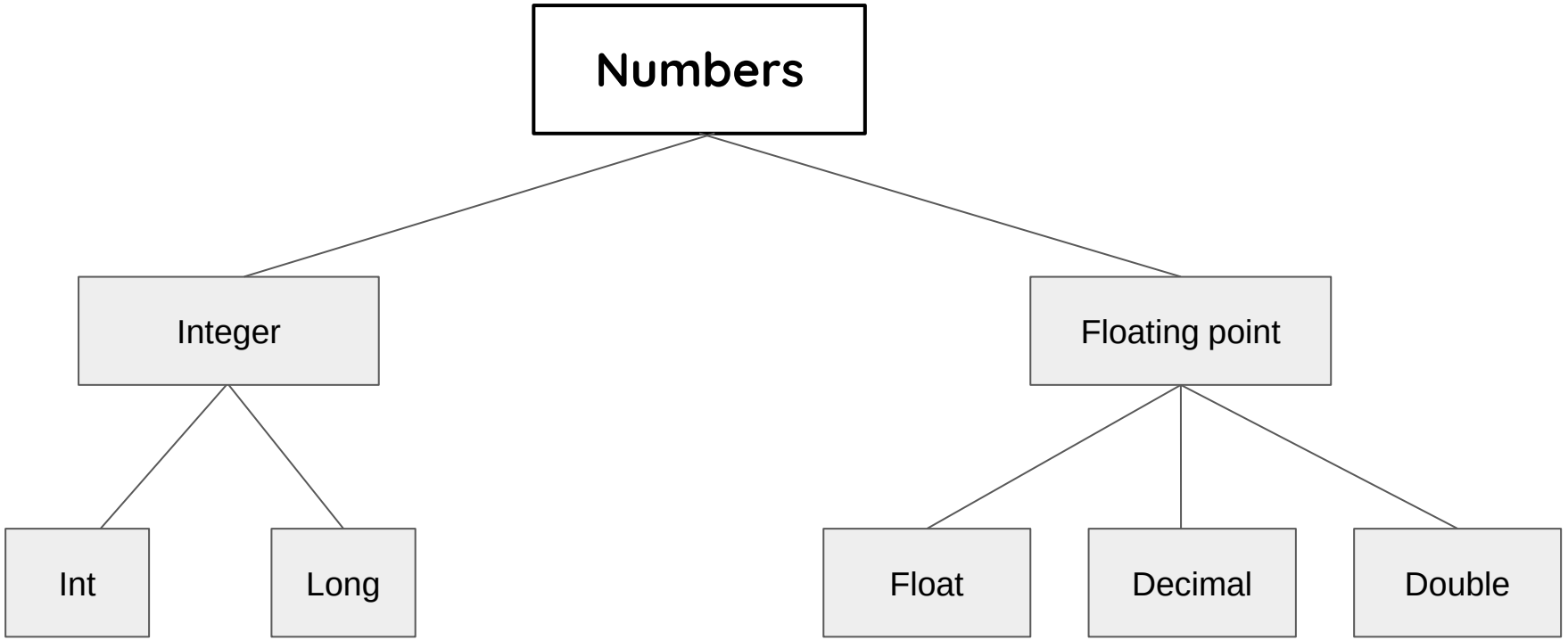
  e.g % $! are not allowed

abstract	as	base	bool	break
byte	case	catch	char	checked
class	const	continue	decimal	default
delegate	do	double	else	enum
event	explicit	extern	false	finally
fixed	float	for	foreach	goto
if	implicit	in	int	interface
internal	is	lock	long	namespace
new	null	object	operator	out
override	params	private	protected	public
readonly	ref	return	sbyte	sealed
short	sizeof	stackalloc	static	string
struct	switch	this	throw	true
try	typeof	uint	ulong	unchecked
unsafe	ushort	using	virtual	void
volatile	while			

Primitive Data Types

They are called primitive data types because they are main built in types and can be used to build other data types.

- Integral numbers
- Real numbers
- Characters
- Booleans


Numbers
Integer
Floating point
Int
Long
Float
Decimal
Double

	Integer	Long
Representation	Int	long
Range/Data size	-2,147,483,648 to 2,147,483,647 (4 bytes)	9,223,372,036,854,775,808 to 9,223,372,036,854,775,807 (8 bytes)
When to use	Representing whole numbers(aka not fractions)	
	• Default option • Conserves more memory	• Storing larger numbers
Example	`int myNum = 100000;`	`long myNum = 15000000000L;` (Note than for long, L must be included at the back of number)

	Double	Decimal	Floating point
Representation	double	decimal	float
Approximate range	±5.0 × 10−324 to ±1.7 × 10308	±1.0 x 10-28 to ±7.9228 x 1028	±1.5 x 10−45 to ±3.4 x 1038
Precision	~15-17 digits	28-29 digits	~6-9 digits
When to use	For fractional numbers		
	Default	• Need more precision • Uses up more space	• Conserves more space • Less precise
Example	`double myNum = 19.99;`	`decimal myNum = 19.99M;` (M is required)	`float myNum = 5.75F;` (F is required)

Characters

	C# Type	.Net Type	Bytes	Range
Character	char	Char	2	Unicode characters

Unicode characters means UTF-16 which includes characters from lowercase a-z and uppercase A-Z.

Boolean

	C# Type	.Net Type	Bytes	Range
Boolean	bool	Boolean	1	True/False

Booleans is a value that represents true or false.

For example,

bool thisvalue = true;

Console.WriteLine(thisvalue);

The programme will display a false

Operators

- Arithmetic
- Comparison
- Assignment
- Logical
- Bitwise

Arithmetic Operations

- Symbols for performing operations are as follows
  - + → Addition
  - - → Subtraction
  - * → Multiplication
  - / → Division
  - % → Modulus
- Rules for order of operations still apply

  E.g: (2+5)*(9/3)=21

  In terms of priority descending, () brackets, * / multiply or divide, + - addition or subtraction.

Comparison Operators

Example

```
var A = 1;

var B = 1;

var C = 3;

Console.WriteLine(A==B);

Console.WriteLine(A<C);
```

Equal	==
Not Equal	!=
Greater than	>
Greater than or Equal to	>=
Smaller than	<
Smaller than or Equal to	<=

Outcome in both examples will be true as A and B both equal to 1, and A is smaller than C

Take note that there is a difference btwn == and =. (covered in next slide)

Assignment Operators

Example

var A = 1;

A+=1; (same as A=A+1)

Console.WriteLine(a);

What will A be?

Answer is A will be 2

Assignment	=
Addition assignment	+=
Subtract assignment	-=
Multiply assignment	*=
Divide assignment	/=

Assignment Operators

What is the difference between == and =?

== is equal. For eg, A==1 means A equals to 1.

But = does not mean equal.

A=1 means we are assigning integer 1 to variable A. It is an assignment.

Logical Operators

This is normally used in boolean
Expressions, which will be used in conditional statements.

For example,
If (a==1 && b==2)
{ a+b; }

&&	And
||	Or
!	Not

So if statement will be covered later. It is a conditional statement. This means that if a equals to 1 AND b equals to 2, if this condition is fulfilled, then the code within this block (which is a+b) will execute. The point here is the && and statement.

Logical Operators

So for the previous example, A must equal to 1 AND B must equal to 2 to execute the if statement. If only A equal to 1 but B is some other value other than 2, or A is some value other than 1 but B is equal to 2 or A is not equal to 1 and B is not equal to 2, the code below the if statement will not run. Both A and B must be true.

For the Or operator, only 1 of the 2 needs to be true to run.

For the Not operator, For example, here this just means if not true, x will add 1.

if (!true)

{x + 1}

Bitwise Operator

This operator is mostly used for windows API or sockets or encryption. We will not be touching too deep on this.

&	And
|	Or

Comments

Comments are when u want to write something in your code but dont want it to be displayed upon execution.

//	Normal comment
/* … */	Multi line comment

For example,

X=1;

//Y = X+1;

Console.WriteLine(Y);

The answer here will be an Error as Y doesnt exist because Y=X+1; statement is not executed as it is a comment.

Non Primitive Data Types

- Strings
- Arrays
- Lists
- Classes(Covered in OOP)
- Enumeration

Strings

- A way of representing a series of characters(no specific limit)
- Default value of string is null if not specified
- Immutable(cannot be altered after creation)
    - Aka `string word= ‘bear’;` 

```
word[0]= ‘p’;
```

Strings

- Can be concatenated(joined together)
  - `string fruit= “apple”;`

`string colour= “red”;`

```
string colouredfruit= colour + fruit;

//red+apple=redapple

Console.WriteLine(colouredfruit);
```

  - Output: redapple

Strings

- String indexing
  - Given that strings are a sequence, indexing is used to refer to a specific position in the sequence
  - Represented as [ ]
  - Note that the first position in a string is 0
  - Index position must be positive

	H	E	L	L	O
Index	0	1	2	3	4

Strings

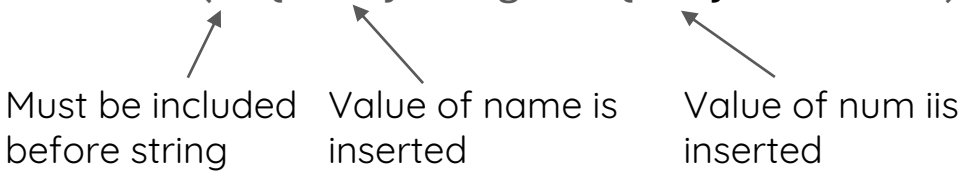
- E.g: You want to print out 4th character in the sequence

```
string name = “Arnold”;

Console.WriteLine(name[3])

Output: o
```

String interpolation

- Substituting values of variables into placeholders in a string
- Means of formatting a string
- E.g

```
string name = “John”

int num = 5

Console.WriteLine($”{name} bought  {num} cookies”)
```

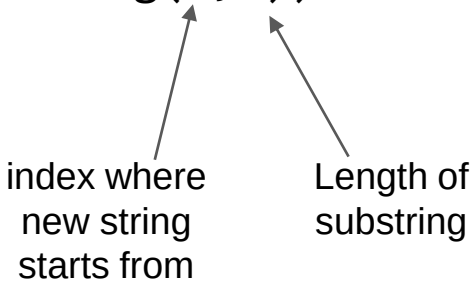



```
Output: John bought 5 cookies
```

Substring

- New shorter string created from the original string

```
string drink = “chocolate milk”

Console.WriteLine(drink.substring(0,9))
```




```
Output: chocolate
```

- Useful when you only want to obtain a certain section of the string

Some Common methods in Strings

Name of method	How it works	How is it declared	Result
Contains()	Checks whether element is present in string	string name = “John” Console.WriteLine(name.contains(“John”))	True
ToLower()	Convert to lower case	string name = “John” Console.WriteLine(name.ToLower())	john
ToUpper()	Convert to upper case	string name = “John” Console.WriteLine(name.ToUpper())	JOHN
IndexOf()	Return index position of element	string name = “John” Console.WriteLine(name.IndexOf(“J”))	0
Trim()	Remove the specified character from the beginning and end of the string	string name = “ John ” Console.WriteLine(name.Trim()) string name = “####John####” Console.WriteLine(name.Trim(“#”))	John

Arrays

- Container for storing multiple values
  - Declared as string[ ], int[ ], double [ ] etc

```
int[] numbers = { 1, 2, 3, 4, 5, 6}
```

Declaring array with these 6 integers

```
int[] numbers = new int[6]
```

Declaring size of array

```
int[] numbers = new int[6] { 1, 2, 3, 4, 5, 6}
```

Alternative syntax for first example (explicitly typed)

- Can be single dimensional, multidimensional or jagged
- Size of array must be specified and cannot be changed(static)

Arrays

- Indexing also works for array

```
var names= {“Terry”, “Jordan” , “Michael”};
Console.WriteLine.(names[0])
Output: Terry
```

Lists

- Similar to arrays, they are another form of storing variables
- Need to declare System.Collections.Generic namespace at start
  I.e using System.Collections.Generic;

Initialising List

```
List<int> numbers = new List<int>();
```

Inserting elements into empty list

```
numbers.Add(1);
numbers.Add(2);
numbers.Add(3);
```

```
numbers--> {1,2,3}
```

Note that you can't print out the list. This is just a representation of what it would look like

Lists

Adding arrays into lists

```
List<int> numbers = new List<int>();
int[] array1 = new int[] {1,2,3};
numbers.AddRange(array1);

Numbers-->{1,2,3}
```

Removing from Lists

```
var fruits = new List<string>{“Apple”, “Banana”, “Orange”}
fruits.Remove(“Banana”)

fruits--> {“Apple”, “Orange”}
```

Lists

Concatenating Lists

```
var names = new List<string>{“Tom”, “Richard”, “Harry”};
var morenames = new List<string>{“Jane”, “Gina”, “Sally”};
names.AddRange(morenames);

Names-->{“Tom”, “Richard”, “Harry”, “Jane”, “Gina”, “Sally”}
```

Lists

Couple of methods for lists

List.sort(): sorts elements in list

```
var names = List<int>{“Tom”, “Zachary”, “Harry”};
names.Sort();

names--> {“Harry”, “Tom”, “Zachary”}
```

List.Reverse(): Reverse the position of elements in list

```
var names = new List<string>{“Jane”, “Gina”, “Sally”};
names.Reverse();
names-->{“Sally’, “Gina”, “Jane”}
```

Lists

List.IndexOf(): Gives index of element

```
var names = new List<string>{“Tom”, “Zachary”, “Harry”}
names.Indexof(“Harry”) --> 2
```

List.Contains(): Checks whether element is in list

```
var stationery = new List<string>{“Pencil”, “Eraser”, “Ruler”}
stationery.Contains(“Pencil”) --> True
stationery.Contains(“Stapler”) --> False
```

Lists vs Arrays

	Array	List
Size of array	Static(Size is fixed)	Dynamic(Able to increase)
Data size	Takes up less memory	Takes up more memory
Memory location	Contiguous	Random
Methods	Fewer	More
Generally, Lists will be used because there will not be space constraints compared to array		

Enumeration

- Also commonly known as enum
- Assigning integer to a group of constants
- Useful when you have a predefined list of values and to make your code more readable
- E.g Suit of card, name of chess pieces

Syntax

```
enum Suit
{
Heart,          //Assigned value of 0
Diamond,        //Assigned value of 1
Club,           //Assigned value of 2
Spade           //Assigned value of 3
}
```

Getting User Input

Getting user Input

- The method for obtaining user input is **Console.ReadLine()**

  Syntax

1)Asking for User input(name in this case)

```
Console.WriteLine(“What is your name”);
```

2)storing User input to name variable

```
string name= Console.ReadLine();
```

3)Printing value of the name variable(Just to check if it works)\\Optional

```
Console.WriteLine($“Your name is {name}”);
```

Getting User Input

- However, there is one problem with this method
- Console.ReadLine() method returns a string so there would be issues if our user input is a number

E.g

```
Console.WriteLine(“What is your age?”);
int age=Console.ReadLine();
Console.WriteLine(“Your age is”+age);

Output:error CS0029: Cannot implicitly convert type 'string' to
'int'
```

Getting User Input

- Fortunately, there is a way around this
- To resolve this issue, we have to convert to string to an integer
- This can be achieved by using the Convert.ToInt32 method(Convert.ToInt16,Convert.ToInt64 will also work)

```
Console.WriteLine(“What is your age?”);
int age= Convert.ToInt32(Console.ReadLine());
Console.WriteLine(“Your age is ”+ age);

Output: Your age is (number you keyed in)
```

Control Flow Statements

Control Flow Statements

- If/Else statements
- Case and Switch statements
- For loop
- While Loop
- Exception handling(Try, catch, finally, throw)

If/Else/Else if

- Statements which will be executed when certain conditions are fulfilled
- Statements will be executed in this order:

  If --> Else if --> Else

If/Else

- If Statement will be executed when conditions are fulfilled

```
int a = 5;
int b = 10;

if (b-a==5)
{
 Console.WriteLine(“The difference is 5”);
}

else
{
Console.WriteLine(“The difference is not 5”);
}
```

Condition

Should include when there is more than 1 statement inside(In this case, it actually doesn’t need to be included)

Executed when b-a equals to 5

Only executed when b-a does not equal to 5

```
Output: The difference is 5
```

If/Else

- Else statements will be executed when the If statement condition is not fulfilled

```
int a = 9;

if (a>=10)
{
Console.WriteLine(“a is greater than or equals to 10);
}

Else
{
Console.WriteLine(“a is less than 10”);
}
```

Output: a is less than 10

Else if

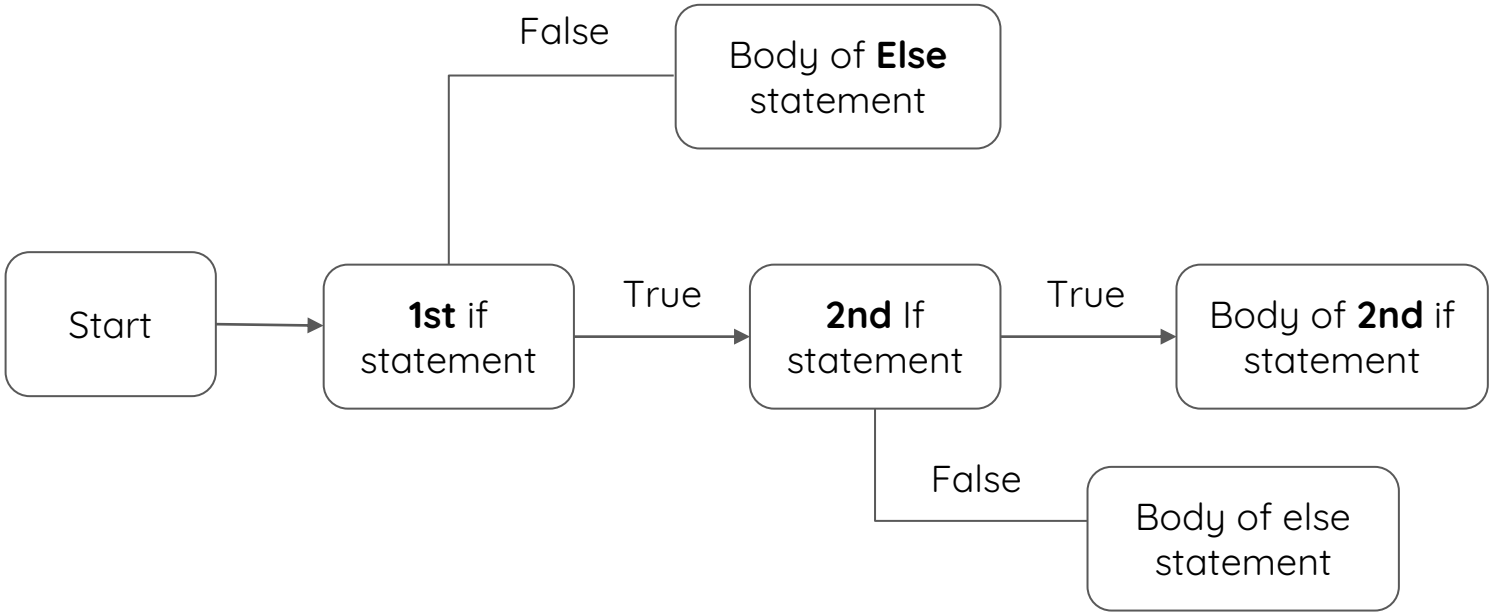
- When if statement is not executed, else if statement will be executed(provided that conditions are fulfilled)

```
int a = 10

if (a>10)
Console.WriteLine(“a is greater than 10”);

else if(a==10)
Console.WriteLine(“a is equal to 10”);

else
Console.WriteLine(“a is less than 10”);

Output: a is equal to 10
```

Nested If statements

- An if statement within an if statement

General Framework of how it works



Nested if statements

```
int a=3; int b=7;
if (a%2==0 && b%2==0)
{
    if (a-b>=0)
    {
    Console.WriteLine(“a is greater than or equal to b and both are even
    numbers”);
    }
    else
    {
    Console.WriteLine(“a is less than b and both are even numbers)
    }
```

In this case, we are trying to investigate if both numbers are even and whether a is greater or less than b

Nested if statements

```
else
{
        if (a-b>=0)
    {
    Console.WriteLine(“a is greater than or equal to b and both are not
    even numbers”);
    }
    else
    {
    Console.WriteLine(“a is less than b and both are not even numbers)
    }
}
```

Case and switch statement

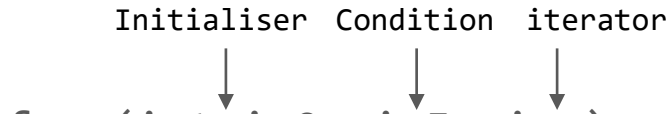
- Efficient way of checking an expression through multiple cases compared to using multiple else if statements

```
int day = 3
switch (day)
{
  Case 1:
         Console.WriteLine(“Monday”);
         break;
  Case 2:
         Console.WriteLine(“Tuesday”);
     break;
  Case 3:
     Console.WriteLine(“Wednesday”);
     break;
  Case 4:
     Console.WriteLine(“Thursday”);
     break;
  Case 5:
     Console.WriteLine(“Friday”);
     break;
   Case 6:
     Console.WriteLine(“Saturday”);
     break;
   Case 7:
     Console.WriteLine(“Sunday”)
     break;
   default:
     Console.WriteLine(“Unknown day”)
}

Output: Wednesday
```

Exit this loop if statement is executed

For loop

- Way of allowing a block of codes to executed repeatedly(Iteration)




```
for (int i=0; i<5; i++)
{
 Console.WriteLine(“Hello World”)
}
```

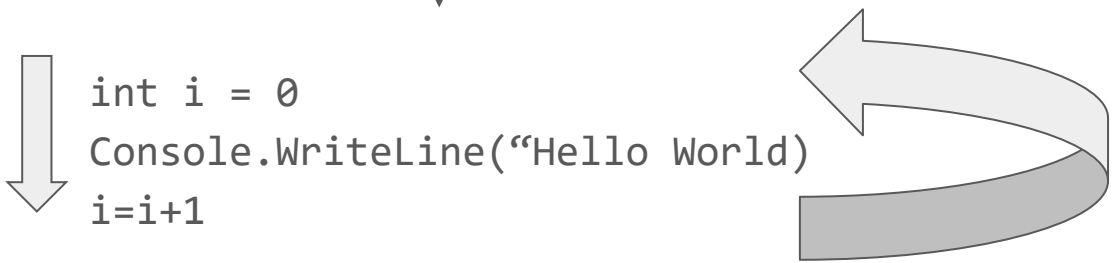
Initialiser: Declare the variable and start the loop

Condition: When it’s not fulfilled, the loop stops

Iterator: Usually it increments(adds) or decrement 1 from the variable in initialiser after each round in the loop

What happens?

```
for (int i=0; i<5; i++)
{
Console.WriteLine(“Hello World”)
}
```



Goes through this cycle until i=5

For loop

- Output: Hello World
  Hello World
  Hello World
  Hello World
  Hello World

Another Example

Let’s say we want to print only odd numbers from 1-11

```
for (int i=1; i<12; i=i+2)
{
 Console.WriteLine(i)
}
```

```
Output: 1
        3
        5
        7
        9
        11
```

1) This time, the initialiser is i=1 as we starting from 1
2) Condition is set to i<12 as we want to stop before 12
3) In order to only include even number, we have to add 2 to skip the odd number

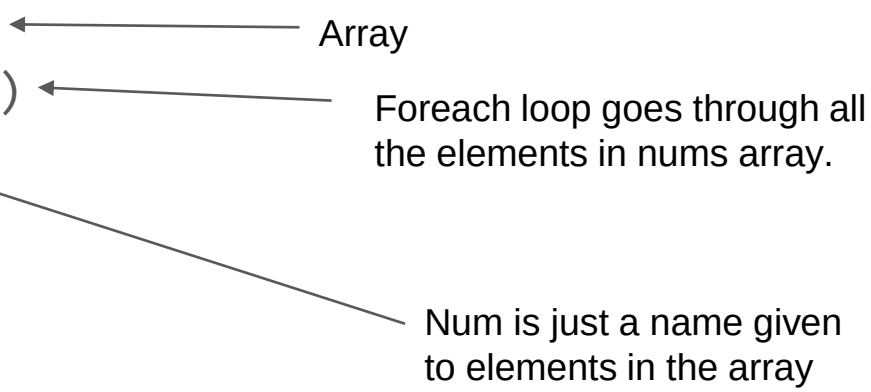
Foreach loop

- Special kind of for loop for arrays, lists etc
- Goes through the elements in the “container”

```
foreach (element in array)
{
 //body of the for loop
}
```

Foreach loop

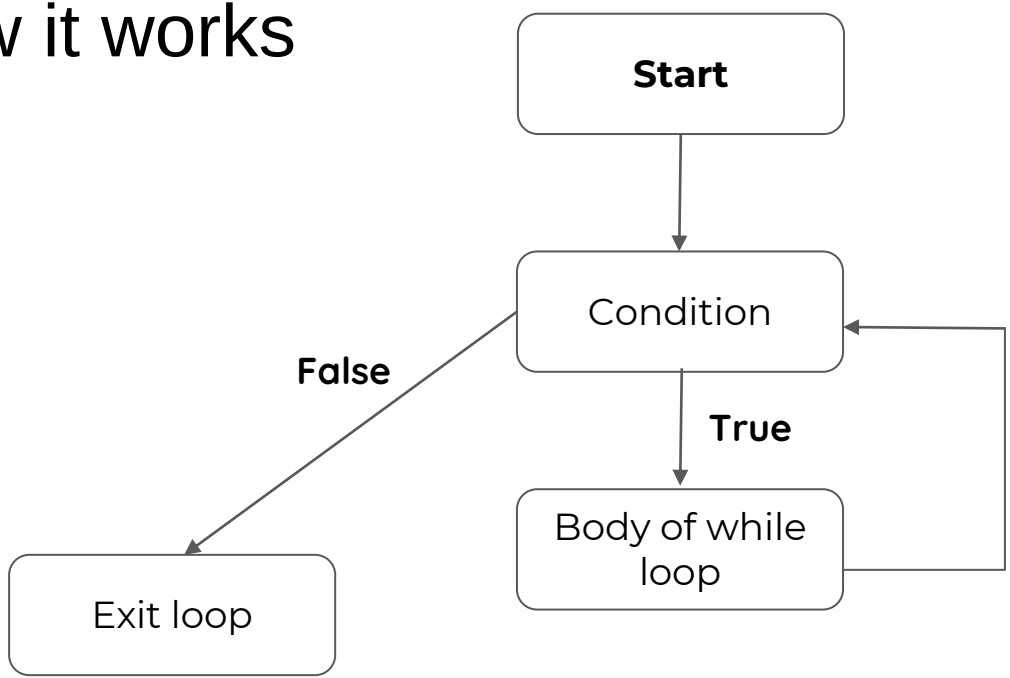
```
int[] nums = {1,2,3,4,5}
foreach (int num in nums)
{
Console.Writeline(num)
}

Output
1
2
3
4
5
```



While loop

- Allow blocks of code to be executed repeatedly when a certain condition is fulfilled
- Can be imagined as a “light switch”
- When the switch is on(condition is fulfilled), the code will be executed
- When the switch is off(Condition is not fulfilled), the code will no longer be executed

How it works



While loop

Syntax

```
int a=0
while( a<5)
{
Console.WriteLine(a);
a++;
}
```

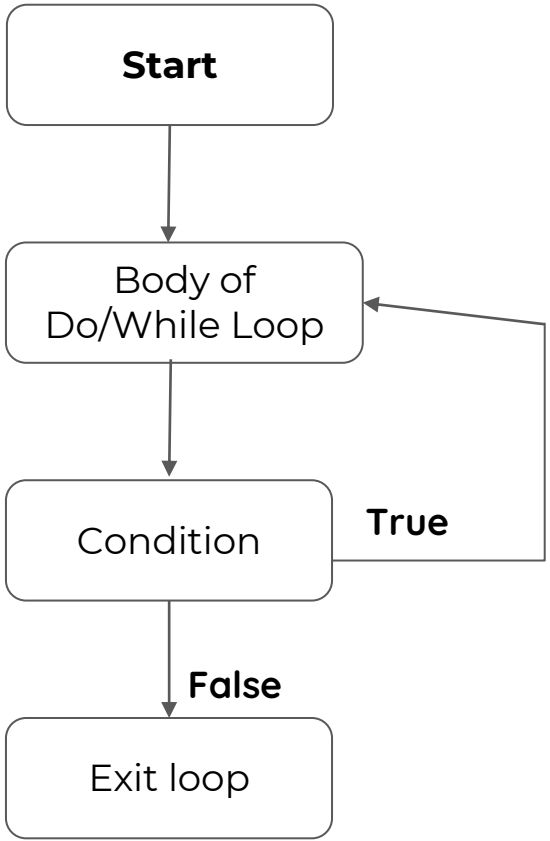
Output

```
0
1
2
3
4
```

Do/While Loop

- Variant of the while loop
- Executed 1 or more times compared to while loop(0 or more times)U
- Useful if you want to executed a block of code before checking the condition

How it works



Do/While Loop

Syntax

```
int b=0
Do
{
Console.WriteLine(“Number is ”+b);
b++;
}
while(b<5)
```

Output

```
Number is 0
Number is 1
Number is 2
Number is 3
Number is 4
```

Break/Continue statements

Break Statement

- Terminate a case in the **switch** statement
- If used in a for/while loop, the loop is terminated and the program will proceed on with the next statement

Continue Statement

- Pushes control of the loop to the next iteration,omitting out any code in the current iteration

Break Statement

Used in for loop

```
for (int i = 0;i<100;i++)
{
    if (i==4)
    {
     break;
    }
 Console.WriteLine(i)
}

Output: 1
        2
        3
```

What is going on?

1)The program goes through the for loop as per normal from i=1 to i=3
2) When i=4, the break statement is initiated

3) The for loop is then terminated and any values from 4 onwards won't be printed

Break Statement

Used in a while loop

```
int i=0
while(i<5)
{
Console.WriteLine(i);
i++;
    if (i==3)
    {
    break;
    }
}
Output: 1
        2
```

What is going on?

While loop will be executed until i=3 and the loop will be terminated after that

Continue Statement

Used in while loop

```
int i=0
while (i<5)
{
i++;
    if (i<3)
    {
    Continue;
    }
Console.WriteLine(i);
}
```

Any code after continue is omitted when condition is fulfilled

```
Output:
4
5
```

1) While loop is executed
2) When i<3, Console.WriteLine() method is not executed
3) When i>=3, Console.WriteLine() is executed

Continue Statement

Used in for loop

```
for (int i=0;i<6;i++)
{
        if (i<4){
        continue;
        }
Console.WriteLine(i);
}
```

Output:4
5

1) For loop is executed
2) When i<4, Console.WriteLine() method is not executed
3) When i>=4, Console.WriteLine() is executed

Error & Exception handling

- In the process of writing code, we may face some errors in our codes
- In order to handle these errors properly, we use these exception handling statements
- Note that these point of these statements is to catch the errors we expect to encounter and to handle them properly

Exception handling Statements:

```
try
catch
finally
throw
```

Error & Exception handling

Syntax

```
try{
//block of code
}
catch (Exception e){
//block of code
}
finally{
//block of code
}
```

Try: Just a block of code to be executed

Catch: Catches the error identified in the try statements and executes another block of code(custom error message)

Finally: Executed regardless of whether there is an error or not

Error & Exception handling

Example:
Division of 2 numbers

```
int result=0;
try{
Console.WriteLine("Enter a number");
int num1= Convert.ToInt32(Console.ReadLine());
Console.WriteLine("Enter another number");
int num2= Convert.ToInt32(Console.ReadLine());
result= num1/num2;
}
catch (DivideByZeroException e){
Console.WriteLine(e.Message);
}
finally{
Console.WriteLine($"The final result is {result}");
}
```

Over here, we are trying to divide number 1 over number 2

When number 2 is equal to 0, the error will be caught and this block of code is executed

Finally statement is then executed to print the result

Throw keyword

- Used to raise exceptions manually
- Useful in forcing exceptions to be introduced when certain conditions are not fulfilled(Refer to example below)

Example

```
//Checking the age of the person that is entering a bar
        int age=0;
        if (age<18){
          throw new ArithmeticException("You must be 18 years old to enter");
        }
        else{
          Console.WriteLine("You may enter the bar");
        }
```

Throw keyword which introduces “Custom” error

Over here, we are introducing an exception which is triggered when the age is less than 18

Exercises

For exercises, we will provide solutions however, these exercises provide you a way to think like a programmer.

Also try explaining every line of code to yourself and identify the problem if you cannot solve a exercise. You shouldnt spend a long time trying to solve one code, usually after 30 minutes you should move on. If you cant solve it take a break!

for some methods of strings etc do note whats the type of its parameter and what type it returns etc etc. These will help you.

Exercises

Note: for all these exercises, ignore input validation unless otherwise mentioned. Assume the user enters a value in the format that the program expects.

For example, if the program expects the user to enter a number, don't worry about validating if the input is a number or not.

When testing your program, simply enter a number.

Exercises(Conditional statements)

1- Write a program and ask the user to enter a number. The number should be between 1 to 3, inclusive of both numbers

If the user enters a valid number, display "Valid number" on the console. Otherwise, display "Invalid number".

Exercises(Conditional statements)

2- Write a program which takes 4 numbers from the console and displays the maximum of the four.

3- Write a program and ask the user to enter the width and height of something like an image for instance. Then tell if this object (for eg image) is in landscape or portrait position.

Exercises(Conditional statements)

4- Your job is to write a program for the ERP speeding camera. Just focus on driving speed of the car

- Write a program that asks the user to enter the speed limit. Once set, the program asks for the speed of a car.
- If the user enters a value less than or equal to the speed limit, program should display “Within speed limit” on the console.
- If the value is above the speed limit, the program should calculate the number of demerit points and display this number.
- For every 5km/hr above the speed limit, 1 demerit points should be added and displayed on the console.
- For example speed limit is 50, from 51-55 is 1 dp, 56-60 is another dp. So display “2 Demerit Points”
- If the number of demerit points is above 12, the program should display “Driving License Suspended”

Exercises (for/while loops)

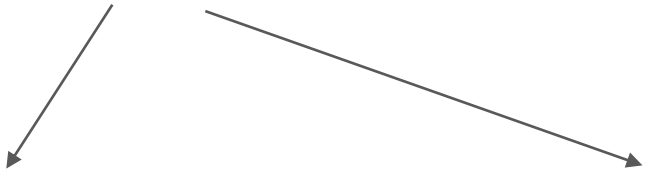
1) Count the number of numbers between 1 and 50 that are divisible by 7. Print out a statement that shows the count.
2) Continuously ask the user to enter a number or “quit” to exit. Print out the sum of all the numbers inputted
3) Ask the user to input a number and print out the factorial of the number. Print out the statement in the following format eg 4!=24

C# Advanced

OOP

Classes

- Building block of an software app
- Anatomy of a class



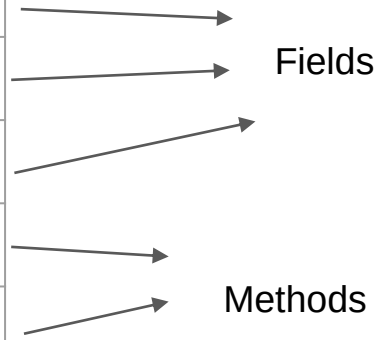
Behaviour which is represented
by Methods/Functions

Data which is represented by FIELDS

Eg of a class

Person

Name:	string
Age:	byte
Height:	float
Walk()	
Talk()	



Instance of a class

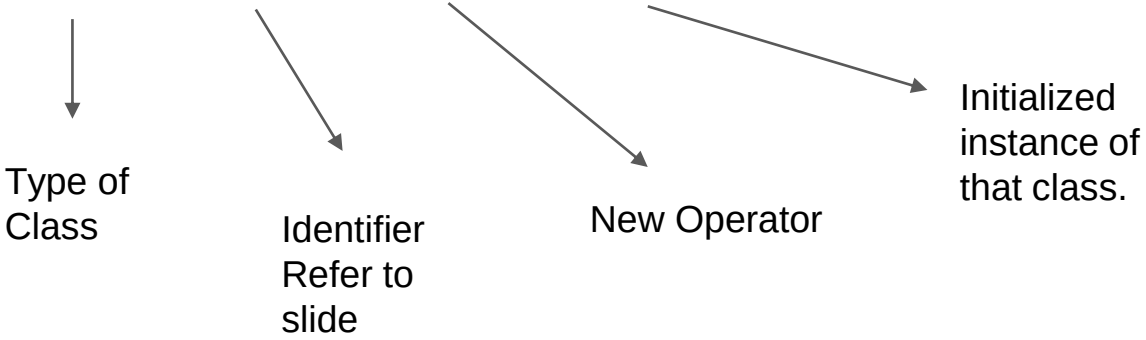
- Instance of a class is called an object
- Person class defines a blueprint from which we can create objects at runtime
- Using the same example, instances of the Person class will be like

Objects
John
Mary
Scott

Creating an Object

This is the way to create a format, using the same example of the Person Class

Person person = new Person();



So here we created an object called person of the Person Class

Note that when we create a Person object it is done in the program.cs page below the main method.

Using objects

- Using the same Person class Example,

person.Name = “John”;

person.Talk();

Class Members

Two types

1. instance : accessible from an object

Eg. var person = new Person();

    person.Talk();

2. static : accessible from class

Eg. console.WriteLine is a static method from the console class

Static members are used to rep concepts tat are singleton, meaning shld only have 1 instance of tat concept in memory like Datetime.Now or Console.Writeline

Declaring static members

```
public class Person

{

    public static int PeopleCount=0;

}
```

Use static keyword. In this example we declared a static field

Creating a class

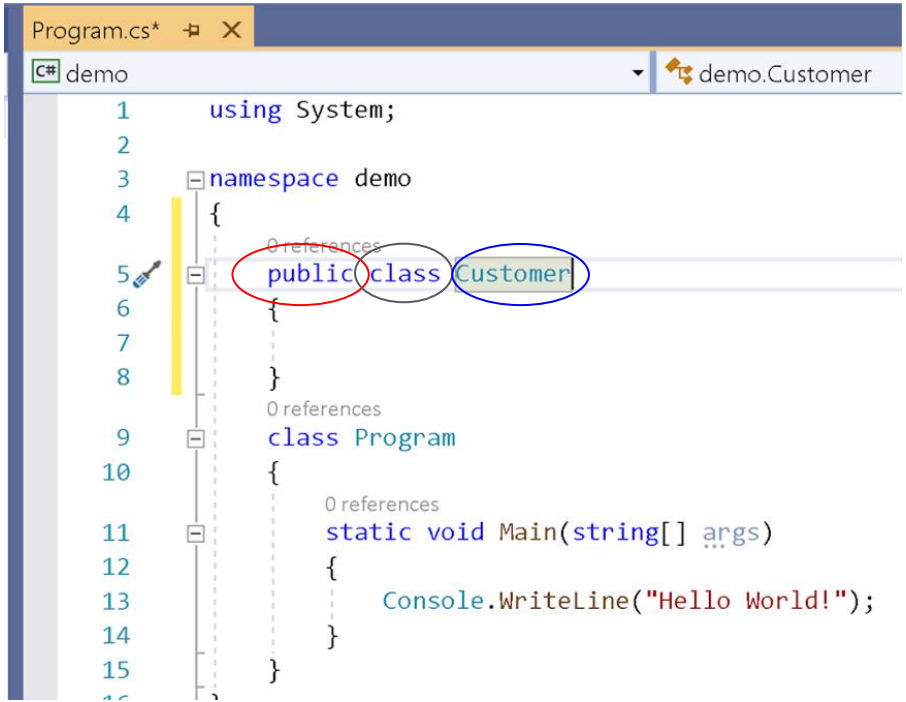
When creating a class, we need to have some keywords.

First is an access modifier (red circle)

Second the class keyword (Black circle)

Last the identifier aka name (blue circle)

Access modifiers will be covered later on



Creating a class

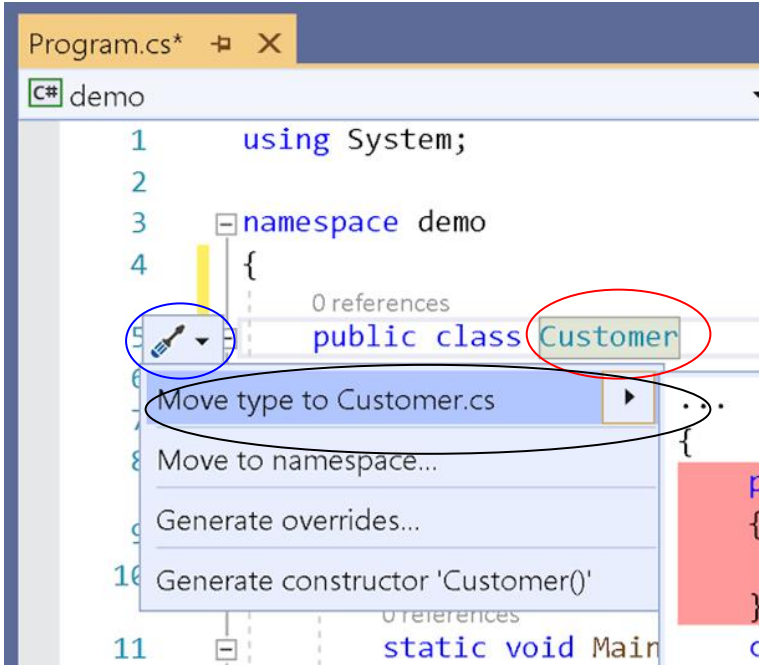
After creating a class in the program.cs page, we need to move the class the another page. To do this

1. We hover our mouse over Customer (red)
2. We click on screwdriver icon (blue)
3. We click on move type to customer.cs

(black circle)

This will bring us to another page called

customer.cs



Constructor

- Is a method that is called when an instance of the class is created

Declaring a constructor

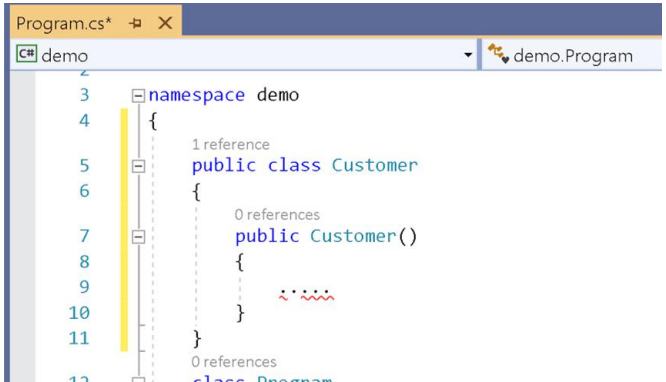
Constructor needs to have the same name as the class. (See image below). Constructor do not need to have a return type not even void.

In the eg, we have a parameterless or a default constructor meaning no parameters. Note if u do not have a constructor defined, c# creates one for u internally, it initialises all fields of the class to default values.

Default values for numbers is set to 0, boolean

types to false and others like strings/arrays are

set to null.



this keyword in Constructor

this keyword, is a keyword that references the current object

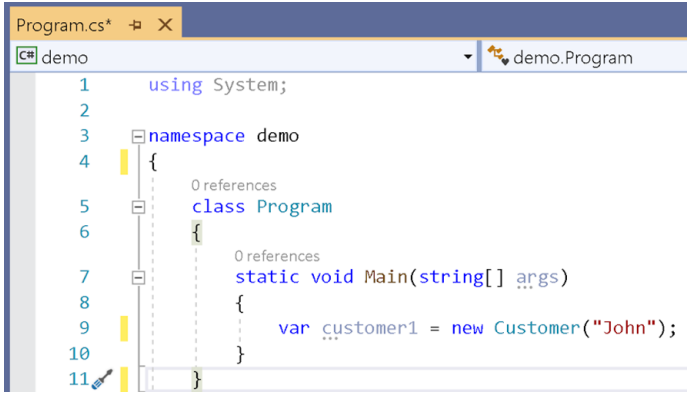
this keyword references to a member of the class in this example it is a field.

So here we have a parameter (we are able to tell as it is within brackets) in the constructor which is a string and the variable is lowerspace name.

this keyword references the field Name (blue circle)

```
1 reference
public class Customer
{
    public string Name;
    0 references
    public Customer(string name)
    {
        this.Name = name;
    }
}
```

this keyword in Constructor



Continuing this example, under program.cs,

Below the main method, we are creating a new

Instance of the class. This instance is called customer1 and we initializing the class through the constructor Customer(“John”) and we pass string “John”.

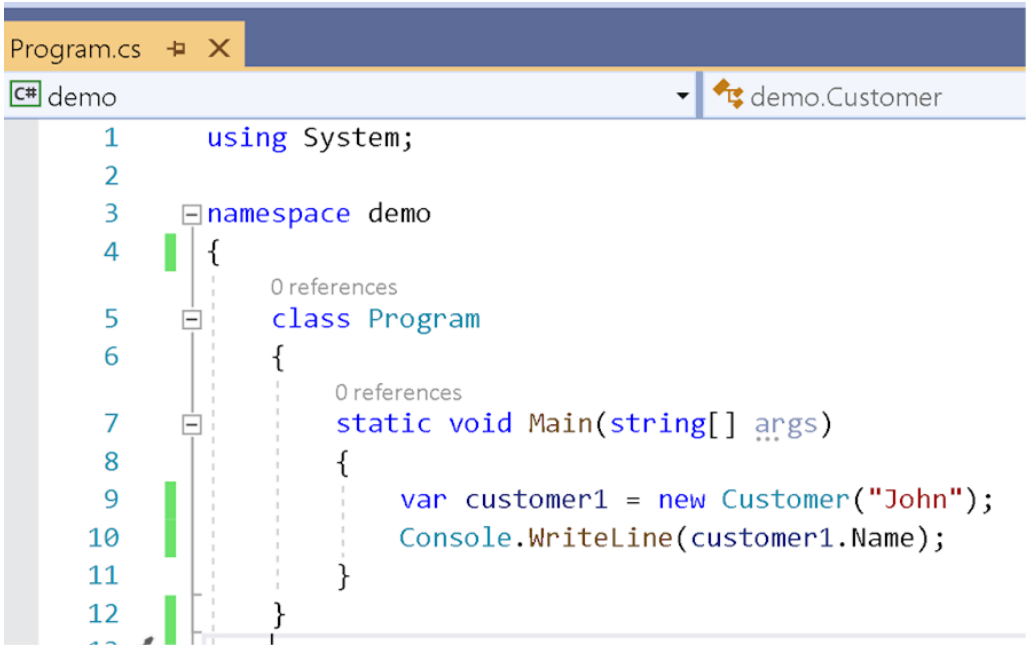
This calls the constructor in Customer.cs in the previous slide.

The constructor runs wtv code is inside it.

The name variable is replaced with “John” so it becomes this.Name = “John”;

this keyword in Constructor

Continuing the example, if i now call the console to display the name field, John will be displayed. Try this! Here we are calling the field Name to be displayed.



Constructor overloading

Overloading means the same name but different signatures.

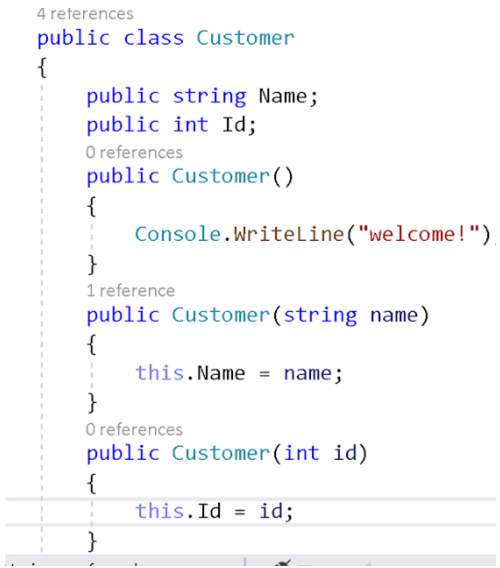
A signature is what uniquely identifies a method that is its return type, its name, its number of parameters.

In this example we have 3 overloads.

One has no parameters

One has a parameter with a string name

One has a parameter with a int id



Constructor Overloading

From this example we see that we have a constructor with no parameters. This means c# will not create a default/parameterless constructor for us, it will use our parameterless constructor here that we defined and when called will execute the Console.Writeline(“welcome”);.

We can also reference to another constructor from one overload, see next slide

Example

So here we have 2 overloads, one which is the

Default constructor with one that prints welcome

The other that prints name.

```
public class Customer
{
    public string Name;
    public int Id;
    1 reference
    public Customer()
    {
        Console.WriteLine("welcome!");
    }
    1 reference
    public Customer(string name)
        :this()
    {
        this.Name = name;
        Console.WriteLine(this.Name);
    }
    1 reference
```

Example

Try executing this code

Execute them individually

Meaning execute customer 1 first

Then add customer 3 inside.

```
class Program
{
    0 references
    static void Main(string[] args)
    {
        var customer1 = new Customer();

        var customer3 = new Customer("John");
    }
}
```

Example

What u see is the first line displays welcome and second and 3rd line displays welcome and John.

This happens due to the this keyword here

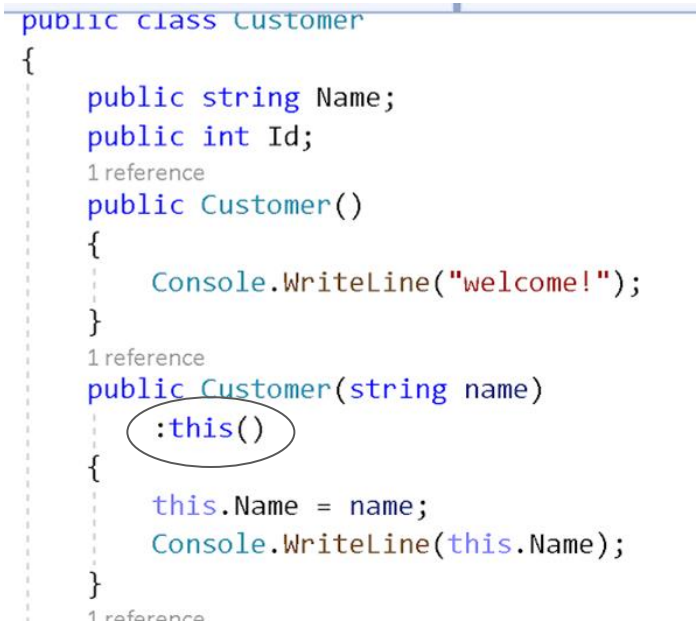
It references to the default/parameterless

Constructor, so when we declare the customer3

Instance, the default constructor is executed first

Followed by the this.Name=name and

console.Writeline(this.Name) line.



```
public class Customer
{
    public string Name;
    public int Id;
    1 reference
    public Customer()
    {
        Console.WriteLine("welcome!");
    }
    1 reference
    public Customer(string name)
        :this()
    {
        this.Name = name;
        Console.WriteLine(this.Name);
    }
    1 reference
```

Methods

A method when called executes whatever code is inside it.

This code inside it will run and can either return an object or have no return object (aka void).

In a method there are

1. signature of the method
2. overloading of a method
3. params modifiers
4. ref modifier and out modifier (these 2 are hardly used in real world application)

Signature of a method

Consists of 2 things.

- number and type of parameters

- name

Example

```
public class TV
{
    1 reference
    public void OnTV()
    {
        Console.WriteLine("TV switched on");
    }
    1 reference
```

For the first method, as shown

Access modifier (blue circle)

Return type of method (red circle)

Name of method (Black circle)

Parameters (pink circle)

This method is public (will be covered later on), the return type is void meaning no return type, the name is OnTV and there are no parameters.

When this method is called, it will execute the code inside it.

Example

```
1 reference
public string Changevolume(int vol)
{
    return string.Format("TV is at volume {0}", vol);
}
0 references
public void ChangeChannel()
```

Access modifier (blue circle)

Return type (red circle)

Name of method (black circle)

Parameters and type (pink circle)

The access modifier is public, the return type is a string, its name is Changevolume and it has 1 parameter which is called vol and the type of vol is an integer.

Example

Try running this code here

First line is

```
class Program
{
    0 references
    static void Main(string[] args)
    {
        var tv1 = new TV();
        tv1.OnTV();
        Console.WriteLine(tv1.Changevolume(100));
    }
}
```

Creating an instance called tv1 of the TV class, the second line is calling the OnTV method which will run the line Console.WriteLine("TV switched on"), this will be the first line displayed on the console. Take note here this method is void. It only runs the code within it but doesnt return anything.

In the 3rd line, we call the changevolume method and give a parameter of 100 which is an integer. It executes whatever code is inside this method. However we have a console.Writeline here wheras the 2nd line doesnt have a console.writeline why is this so?

Example

This is because this changevolume method returns a string. This string will not be displayed on the console without us writing console.writeline(tv.changevolume(100));.

Try writing tv.changevolume(100); on the visual studios and see if anything is displayed on the console.

Wheras for OnTV method, console.writeline is executed within the method itself.

Take note that changevolume returns a string while ontv doesnt return anything.

Method Overloading

- Similar to constructor overloading, we will be overloading our methods now instead
- This means that **multiple methods** in the same class can have the **same name** but **different parameters**(different number and type)
- In the next example, we will create 3 different methods with the same name(addnumbers)

1) 1st method parameter takes in 2 int and it returns an int
2) 2nd method parameter takes in 3 int and also returns an int
3) 3rd method parameter takes in 2 double and returns a double instead

Method Overloading

```
Method 1
public int addnumbers(int a, int b){
return a+b;
}

Method 2
public int addnumbers(int a, int b, int c){
return a+b+c;
}

Method 3
public double addnumbers(double a, double b){
return a+b;
}
```

1st method takes in 2 arguments(aka 2 numbers) and return the sum of both numbers

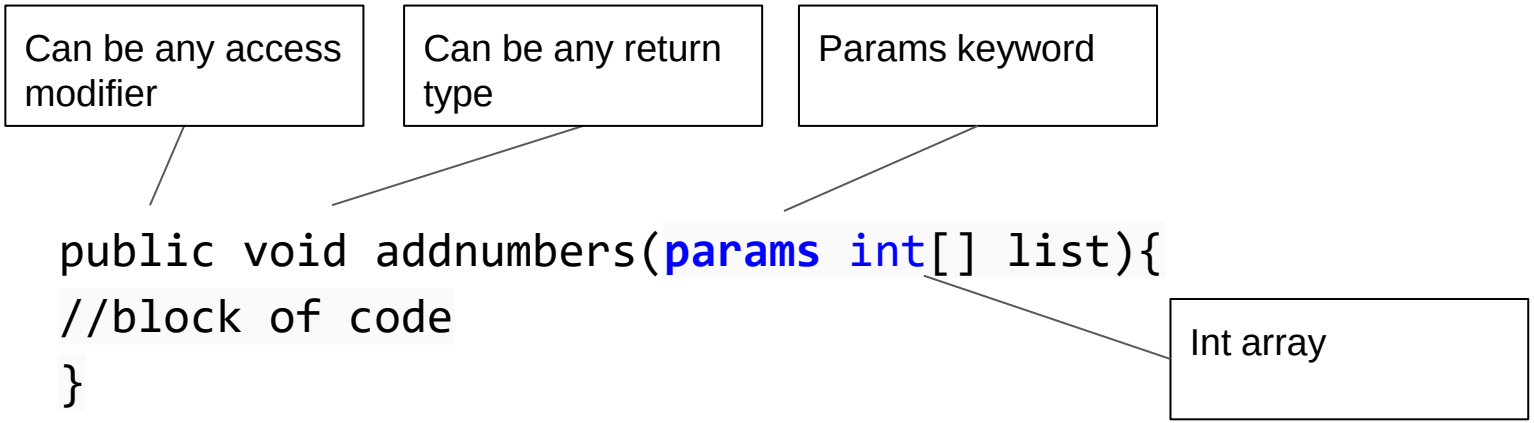
2)2nd method has the same name but take in 3 arguments(numbers) and returns the sum of all 3 numbers

3rd method has the same name but takes in 2 arguments of different data type(double) and returns the sum(which is a double instead of int now)

Param keyword

- Used when you want to take in variable number of arguments(unknown amount)
- Parameter must be of single-dimensioned array data type(e.g int[]. string[])

Syntax



Example

```
public static void Addnumbers(params int[] list)
    {
        int total=0;
        foreach(var i in list)
        {
          total+=i;
        }
        Console.WriteLine(total);
    }
```

Adding a list of numbers together

1) We are creating a method which allows us to add up a variable number of numbers(can be 0,1,2,3 etc)
2) **Params** keyword allows us to add a variable number of numbers
3) We are doing a basic foreach loop to add all the numbers in the array
4) We use the Console.WriteLine Method to print the total sum

Invoking our method

```
//There are several ways that we can input our parameters

static void Main(string[] args)
{
// Method 1 (A list of numbers that are spliced by commas)
Addnumbers(1,2,3,4);
//Method 2(An int array)
int[] intarray= {1,2,3,4};
Addnumbers(intarray);
//Either way, both methods will give the same answer when invoked in Main Method)
}
```

Output:10

4 Key Principles of OOP

1) Abstraction
2) Encapsulation
3) Inheritance
4) Polymorphism

Encapsulation

- Bundling data(fields) with the methods which will work on the data
- This means that the fields in a class are hidden from other classes and can only be accessed via methods
- Used for information hiding(isolating certain information so as to restrict access to this information by the user)
- Primary means of encapsulating information can be through **“getters”** and **“setters”**

Getters: Retrieving data from the field
Setters: Assigning a value to the field

Encapsulation

How to encapsulate data?

1) Declare all fields in the class as private
2) Use public methods such as get and set methods(or constructor) to access and alter the value stored in the field

*Note that you should not use get and set methods for every field as this will essentially expose the internal details and render encapsulation pointless

```
public class Child
    {
        private int age;
        private string name;

//Get method to access the age
        public int Getage() {return age; }
//Set method to assign a value to the age
        public void Setage(int age) {this.age=age;}
//Get method to obtain the name
        public string Getname() {return name;}
//Set method to assign name a value
        public void Setname(string name)
                            {this.name=name;}

    }
```

Fields declared as private

Encapsulation

```
/*Let's say that we want to assign a child called Tim with the following properties(age:7, name:Timothy)*/
```

```
var Tim=new Child();
Tim.Setage(5);
Tim.Setname("Timothy");
```

1)We create an object called Tim
2)We use the set methods to assign values to name and age

```
Console.WriteLine(Tim.Getage());
Console.WriteLine(Tim.Getname());
```

We invoke the Console.WriteLine method here to see the value returned by the Get Methods

Encapsulation

- The constructor can also be used in assigning a value to the fields

Using the earlier example

Instead of

```
public void Setage(int age) {this.age=age;}
public void Setname(string name){this.name=name;}
```

We can use the constructor to combine both set methods

```
public Child(int age, string name)
{
this.age=age;
this.name=name;
}
```

Encapsulation

- Another more concise way of presenting get and set methods is through auto-implemented properties
- How it works is that the compiler will create a private field which can only be accessed through ***get*** and ***set*** accessors

Syntax

```
public int Age { get; set;}
```

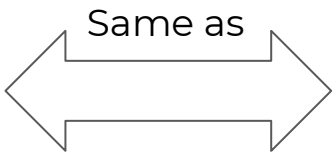
Get accessor which is same as get method from earlier example

Set accessor which is same as set method from earlier example

Encapsulation

```
public int Age { get; set;}
```

Same as



```
private int Age;

public int getage()
{
return Age;
}

Public void setage(int age)
{
this.Age=age;
}
```

Note that it is public here to allow the get and set accessors to be public

The field “Age” will be private

Encapsulation

```
//Printing out the age of Tim(Class in this example is Child)
```

Get and Set accessors

```
var Tim=Child();
Console.WriteLine(Tim.Age=10);
```

What is happening is that Tim.Age is assigned a value of 10(set) and then afterwards that value of 10 is returned(get)

Get and Set Method

```
var Tim=Child();
Tim.setage(10);
Console.WriteLine(Tim.getage());
```

```
Output:10
```

Encapsulation

- Fields can be made read-only if you use only the get accessors
- Purpose of this is to restrict any form of modification to the fields

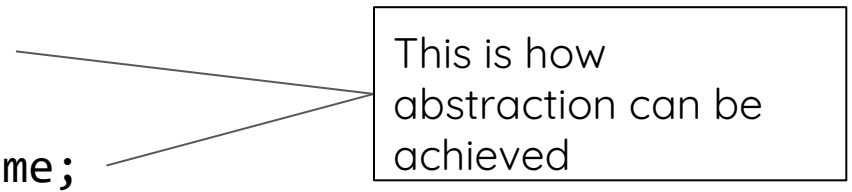
Example

```
public int Age { get; }
```

Abstraction

- Used together with encapsulation to achieve the same purpose- information hiding
- Hiding all the unnecessary details and only revealing data that is essential
- Most commonly achieved by declaring all fields as private(shown earlier in Encapsulation portion)

```
private int age;

private string name;
```

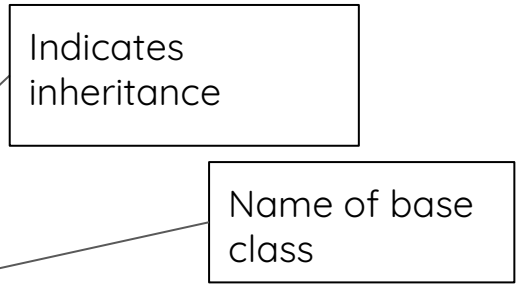


Inheritance

- As the name suggests, the derived class(child) inherits fields and methods from the base class(parent)

Syntax

```
class Parent{
//fields
//methods
}

class child : Parent{
//fields
//methods
}
```



Inheritance allows many new classes which share the same fields and properties to be created from one base class(parent)

Inheritance

- Mostly used for code reusability

Example: Creating a base class called “Parent”

```
//Base class
 public class Parent{
      public void Talk()
      {
        Console.WriteLine("Talking");
      }
      public void Walk()
      {
          Console.WriteLine("Walking");
      }
    }
```

In this case, the parent class has 2 methods Talk() and Walk()

Inheritance

```
//Derived class
 public class Child:Parent
   {
     public void Cry()
     {
      Console.WriteLine(“Crying”)
     }
   }
```

The child class will inherit the 2 methods(don’t need to state again)from earlier plus have a new method called Cry()

Inheritance

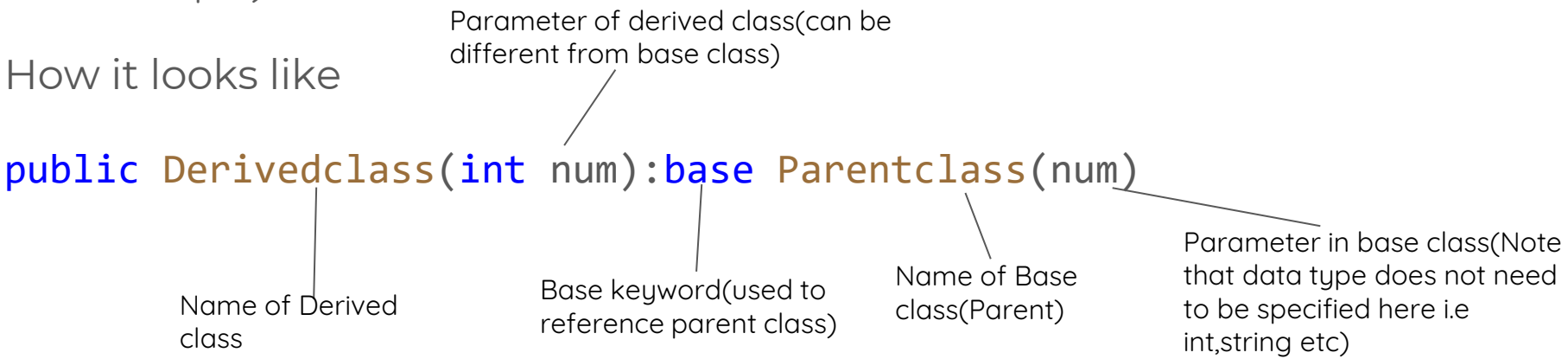
```
 static void Main(string[] args)
{
//Testing out methods that ‘Child’ class has
var Tim= new Child();
Tim.Talk();
Tim.Walk();
Tim.Cry();

Output:
Talking
Walking
Crying
```

Inheritance

Declaring Constructors for Derived class

- For the derived class(child), a colon(:) has to be placed after the new constructor created and the base keyword together with the original constructor will be placed after the new constructor(Look at below example)

How it looks like



Inheritance

Example
Inheritance from Animal to Cat Class

```
//Base Class
public class Animal
    {
      public int Age{get; set;}
      public string Breed{get; set;}

      public Animal(int age, string breed)
      {
        this.Age=age;
        this.Breed=breed;
      }
    }
```

What we are doing is creating a class called Animal and using a constructor to assign values to the fields “Age” and “Breed”

Inheritance

```
//Derived Class
public class Cat:Animal
    {
      public string Owner{get; set;}

      public Cat(int age,string breed, string owner):base(age,breed)
      {
        this.Age=age;
        this.Breed=breed;
        this.Owner=owner;

      }
    }
```

In this new derived class, we have

1. Created a new field/property called Owner
2. Constructed a new constructor to take in input for owner, breed and age(base class)

base: Indicates reference to Base class(parent class)

Original parameters also have to be keyed in for the base constructor(without the data type)

Polymorphism

- Used in tandem with inheritance
- Enables derived classes to implement the same properties and methods in a different way
- The way to do this is with the **virtual** and **override** keyword

Syntax

```
//base class
public virtual void Method()
{//block of code}
```

```
//Derived class
public override void Method()
{//block of code}
```

Polymorphism

Example
Creating a new class “Cow” that will make a different sound

```
//base class

public class Animal
{
 public virtual void Makesound(){
 Console.WriteLine(“Animal Sound”)
 }
}
```

The method “MakeSound” has the keyword **virtual** to indicate that derived classes(children) can implement the same method differently

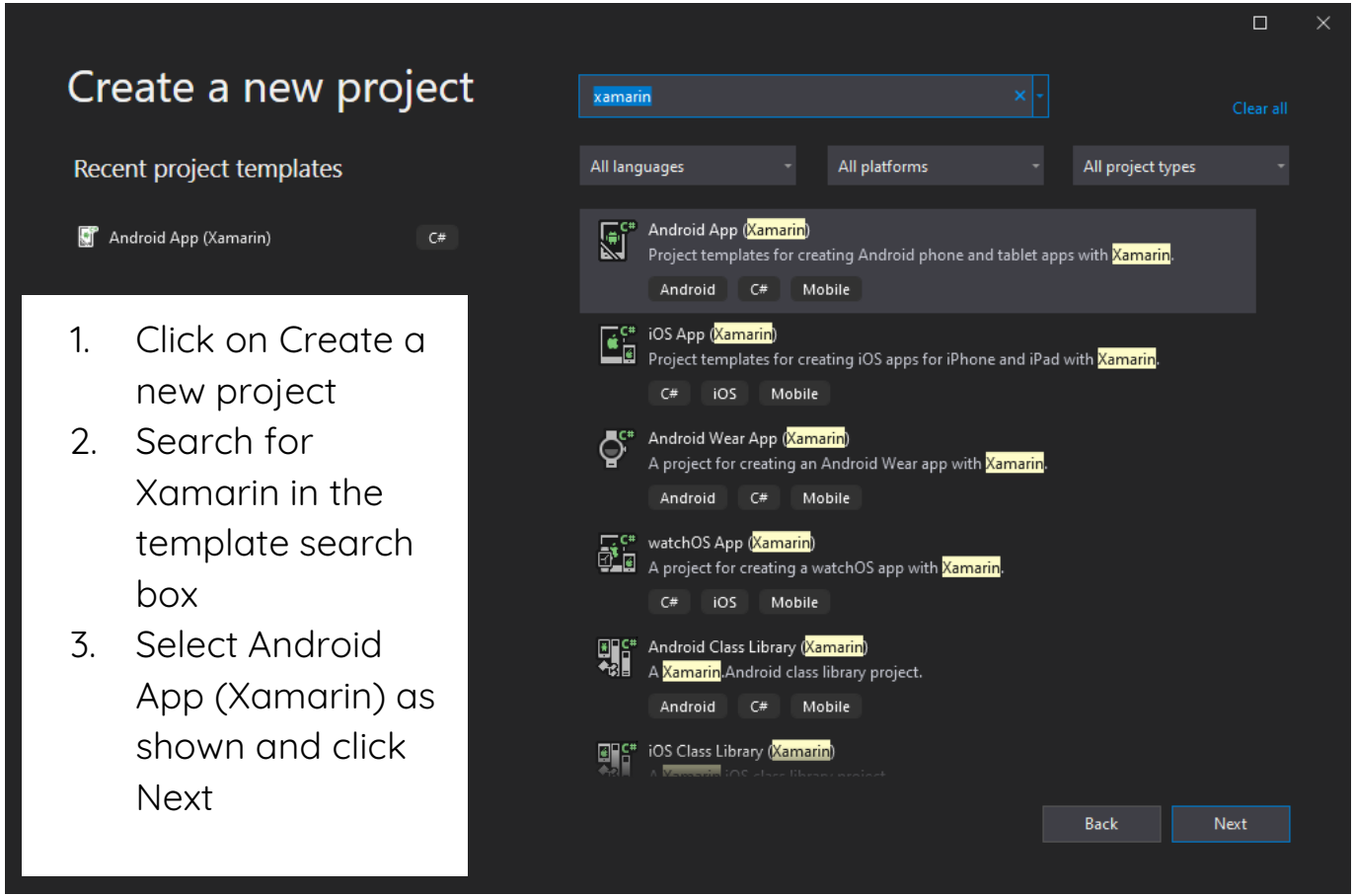
```
//Derived class

public class Cow : Animal
{
 public override void MakeSound(){
Console.WriteLine(“Moo”)
 }
}
```

The keyword “override” allows the method “MakeSound” to adopt a new implementation (`Console.WriteLine(“Moo”)` in this case)

Contents

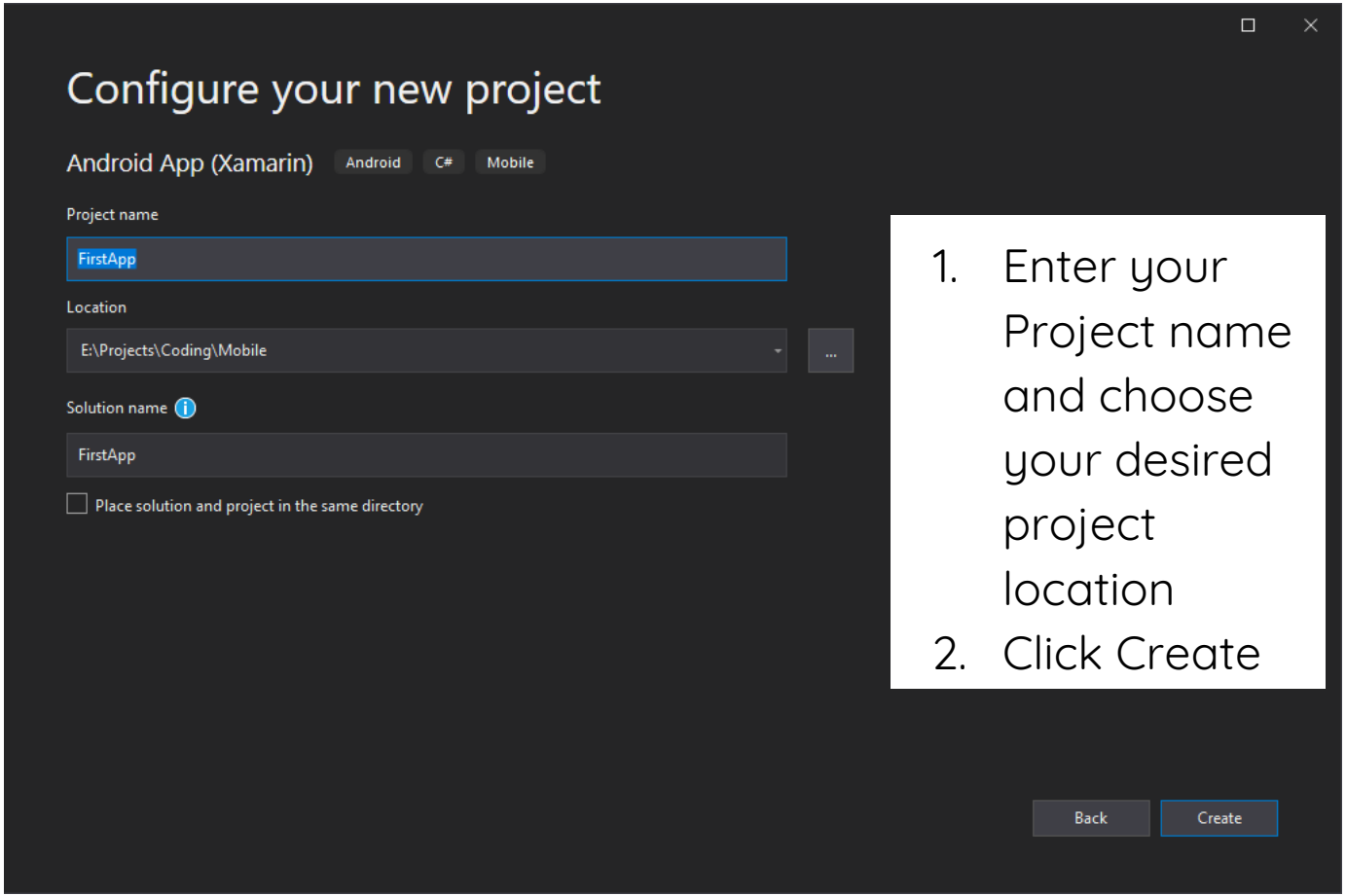
Creating your first Android Application (Windows)

Creating your first Android application (Windows)

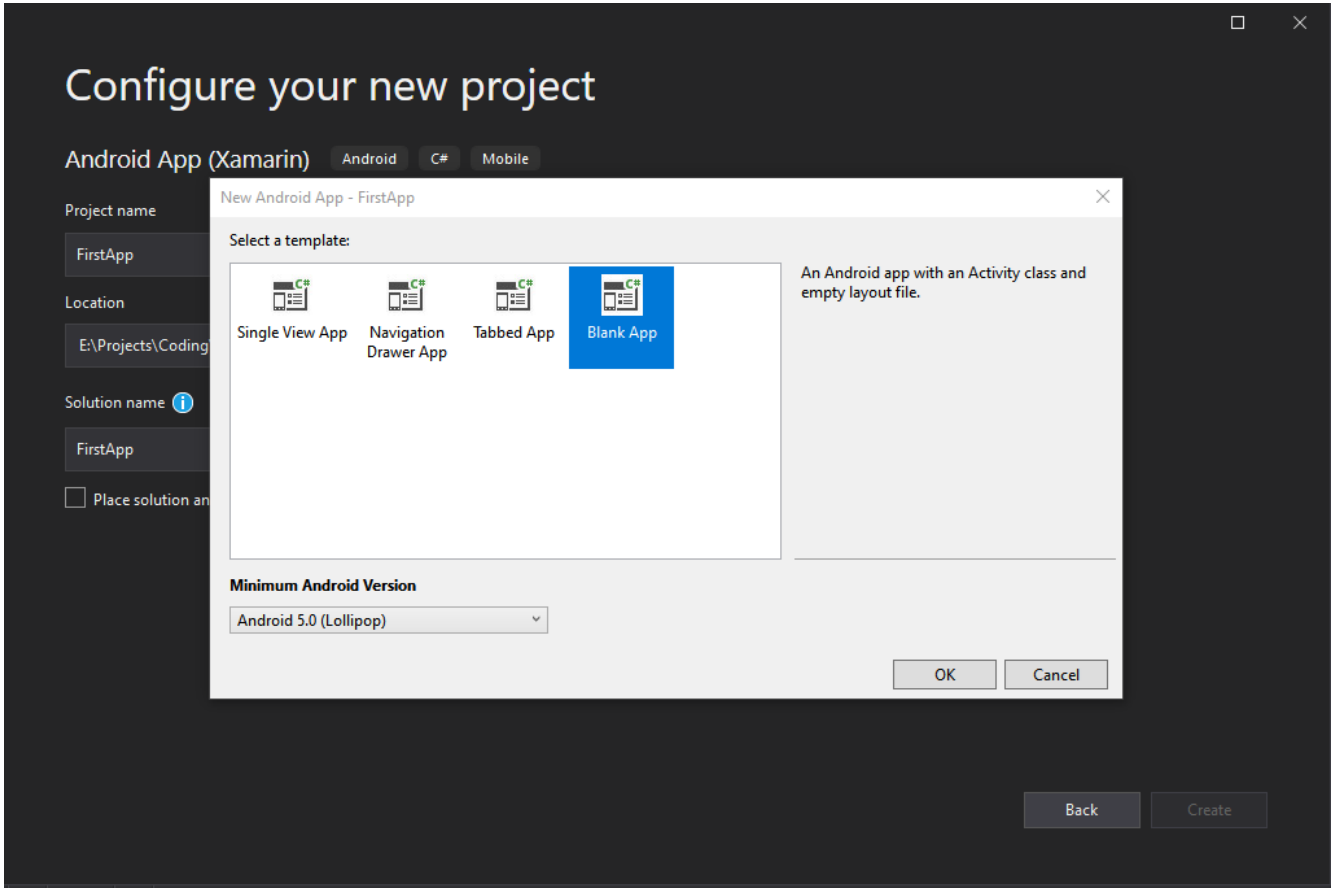


1. Click on Create a new project
2. Search for Xamarin in the template search box
3. Select Android App (Xamarin) as shown and click Next

Creating your first Android application (Windows)



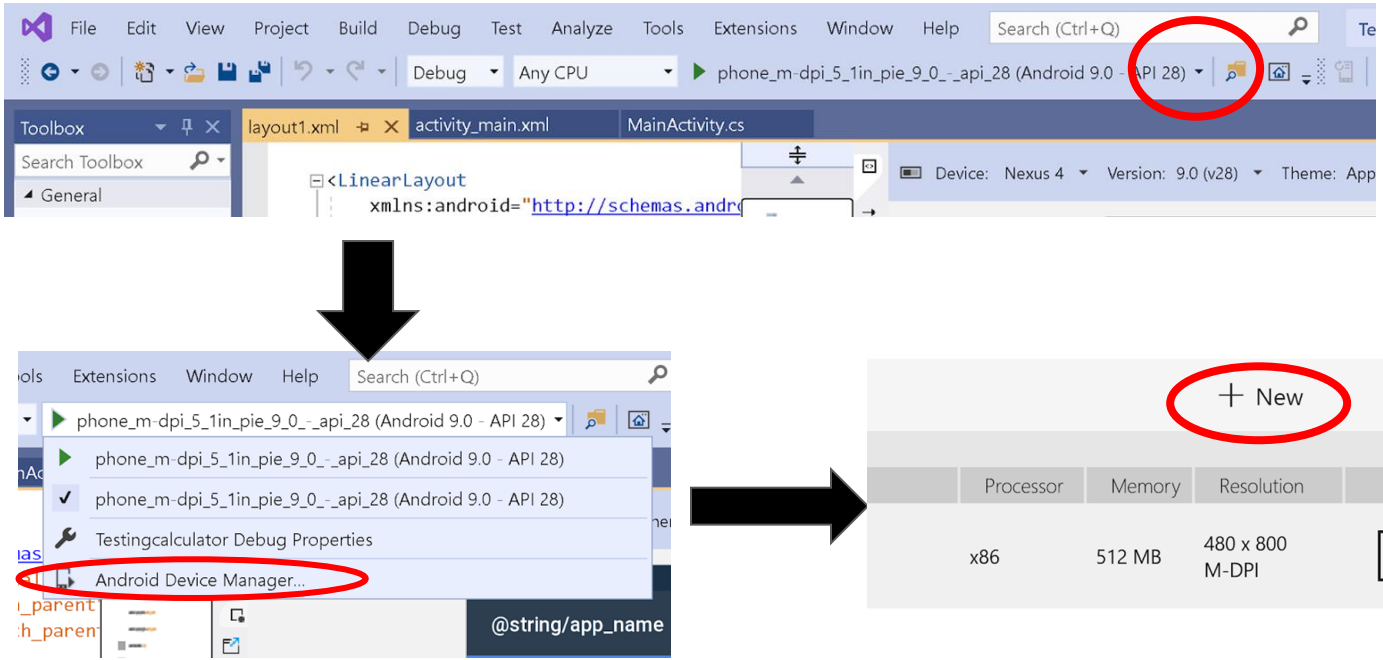
Creating your first Android application (Windows)



1. Select Blank App
2. Choose your Minimum Android Version
3. Click OK

*Better to choose older Android versions
so that older devices can be supported

Android Emulator



Android Emulator

Can rename ur device

-Processor choose x86 for

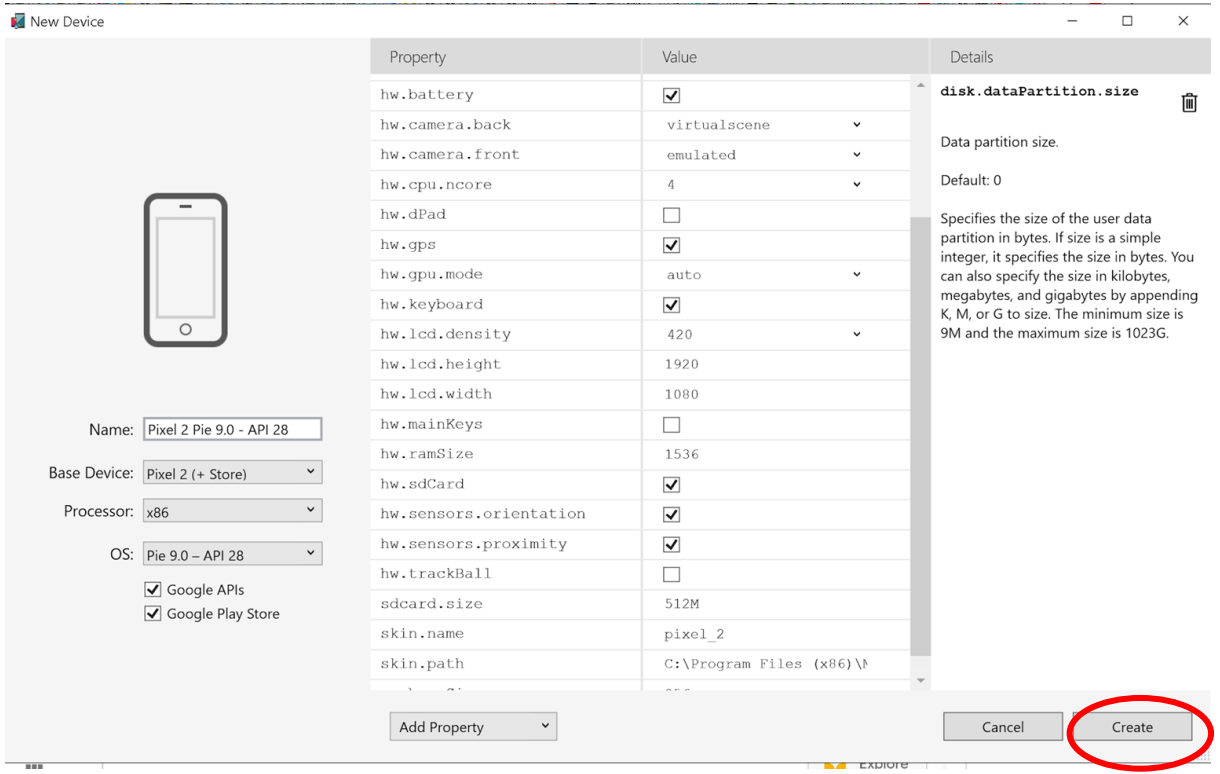
Faster speed

-Base Device can choose

Diff phone models

-OS choose one that is supported

Click on create once done



Creating your first Android Application (Mac)

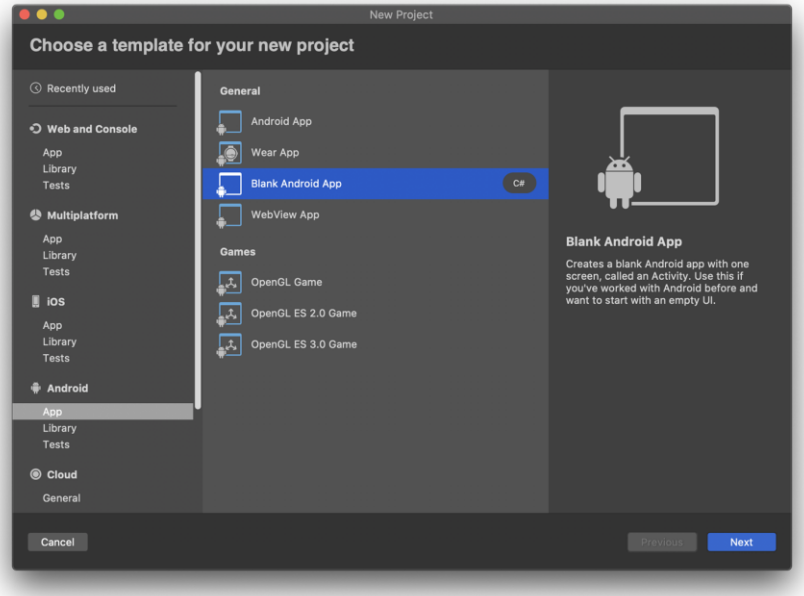
Creating your first Android Application (Mac)

Click on New

Select App under the Android section

Choose Blank Android App

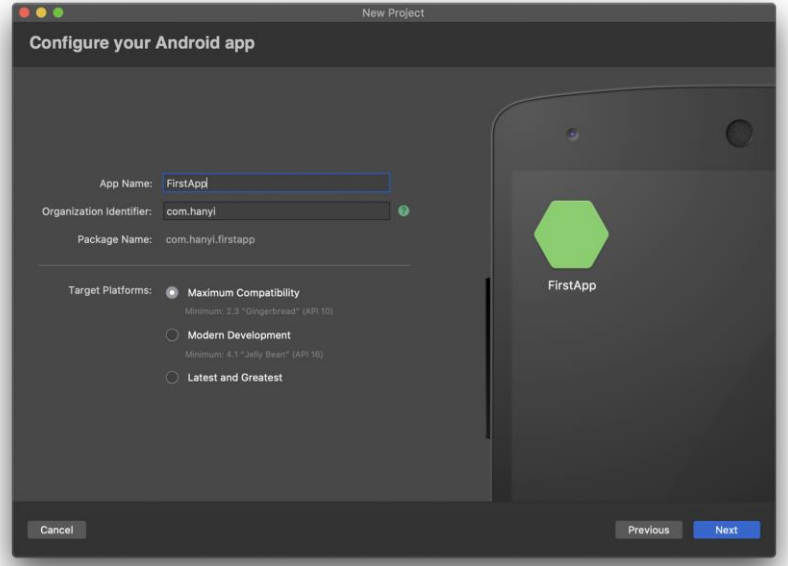
Click Next



Creating your first Android Application (Mac)

Enter your App Name

Click Next

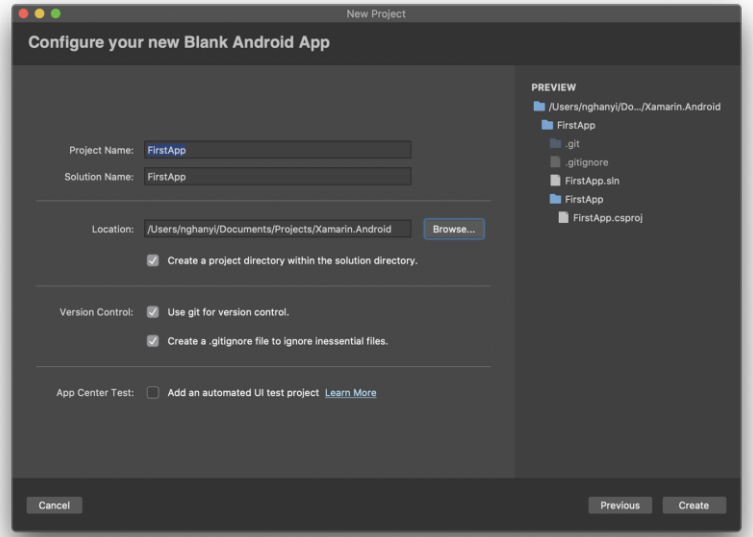


Creating your first Android Application (Mac)

Enter your Project Name

Choose your desired project location

Click on Create

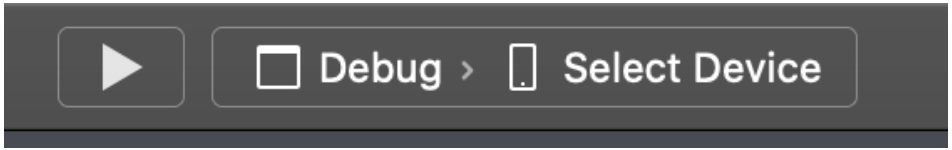


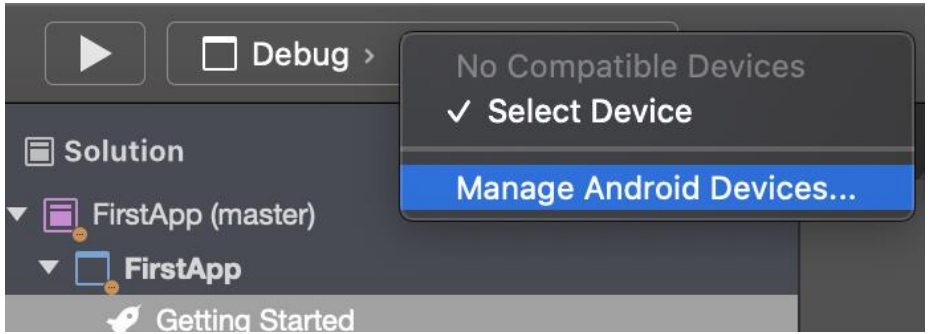
Android Emulator (Mac)

Click Select Device on top left

Select Manage Android Devices

Click on + New Device





Android Emulator (Mac)

Can rename ur device

-Processor choose x86 for

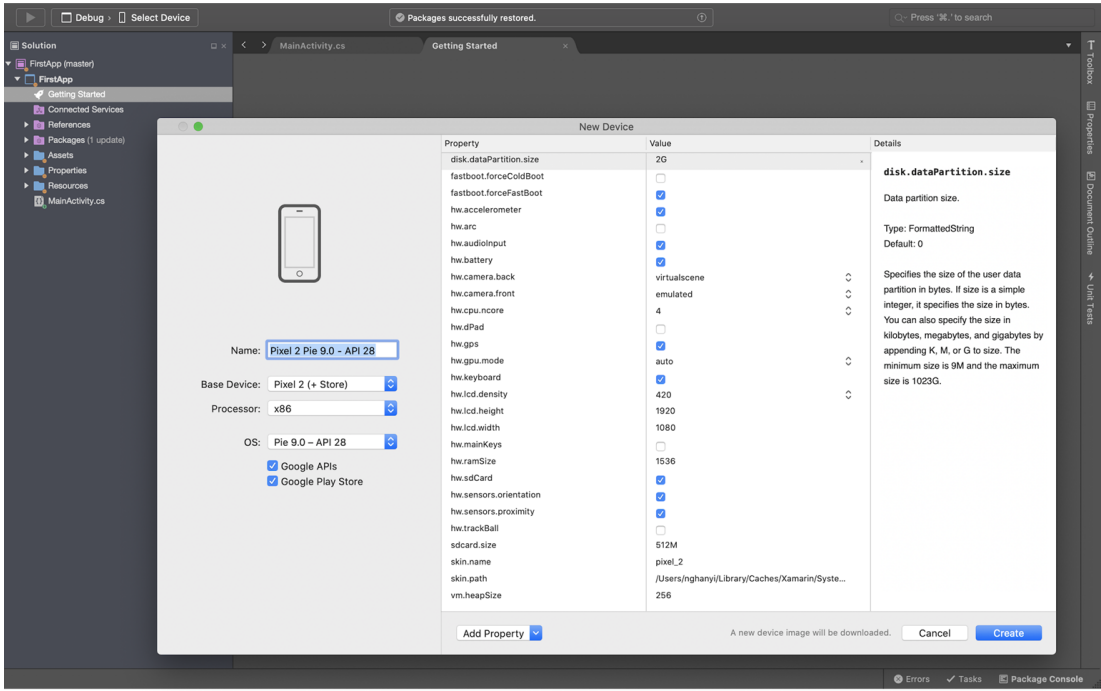
Faster speed

-Base Device can choose

Diff phone models

-OS choose one that is supported

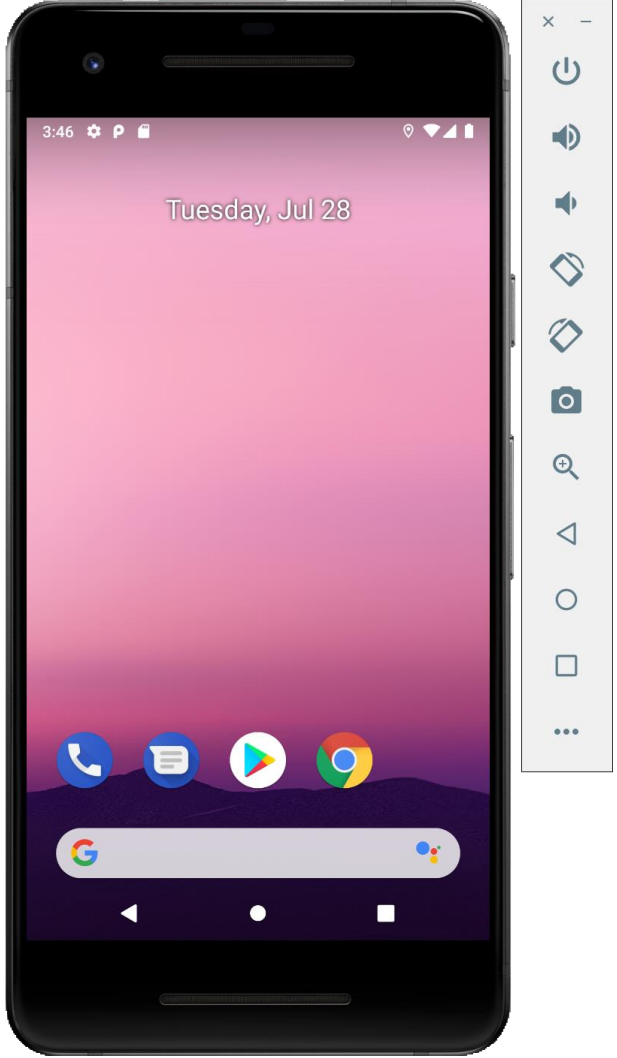
Click on Create once done



Android Emulator (Mac)

Click on Play

Make sure that your emulator is working



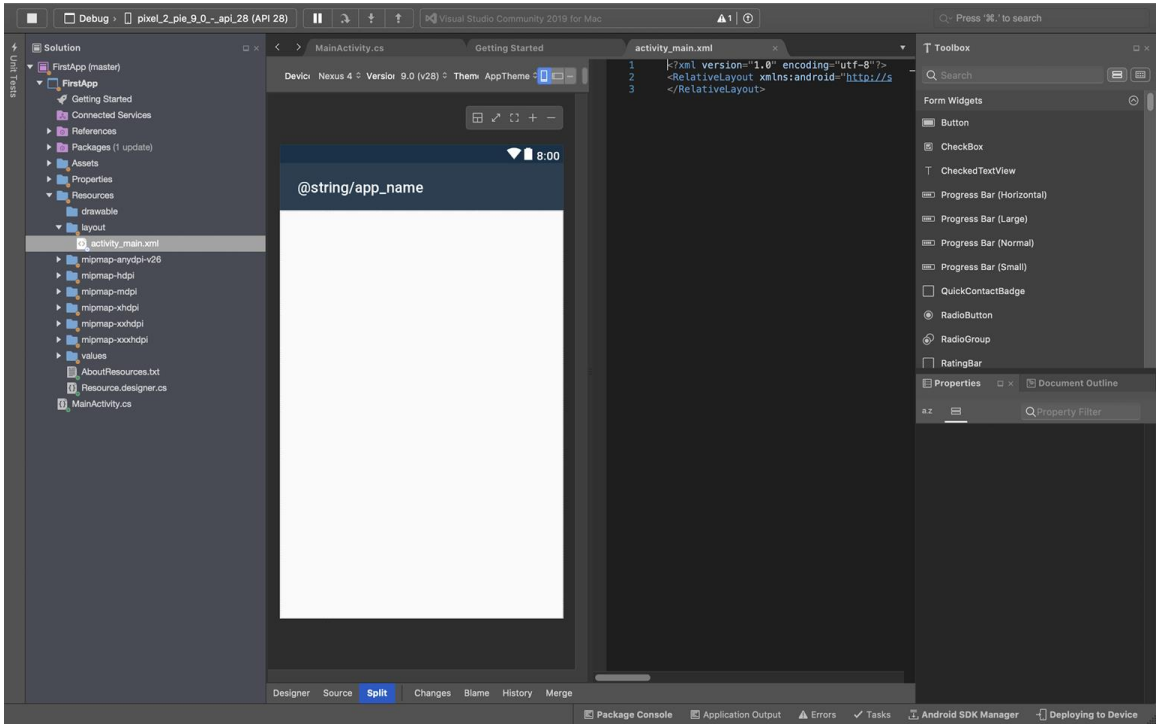
Introduction

Same for both Windows and Mac

The layout (or screen) for the app can be found under:
Resources > layout > activity_main.xml

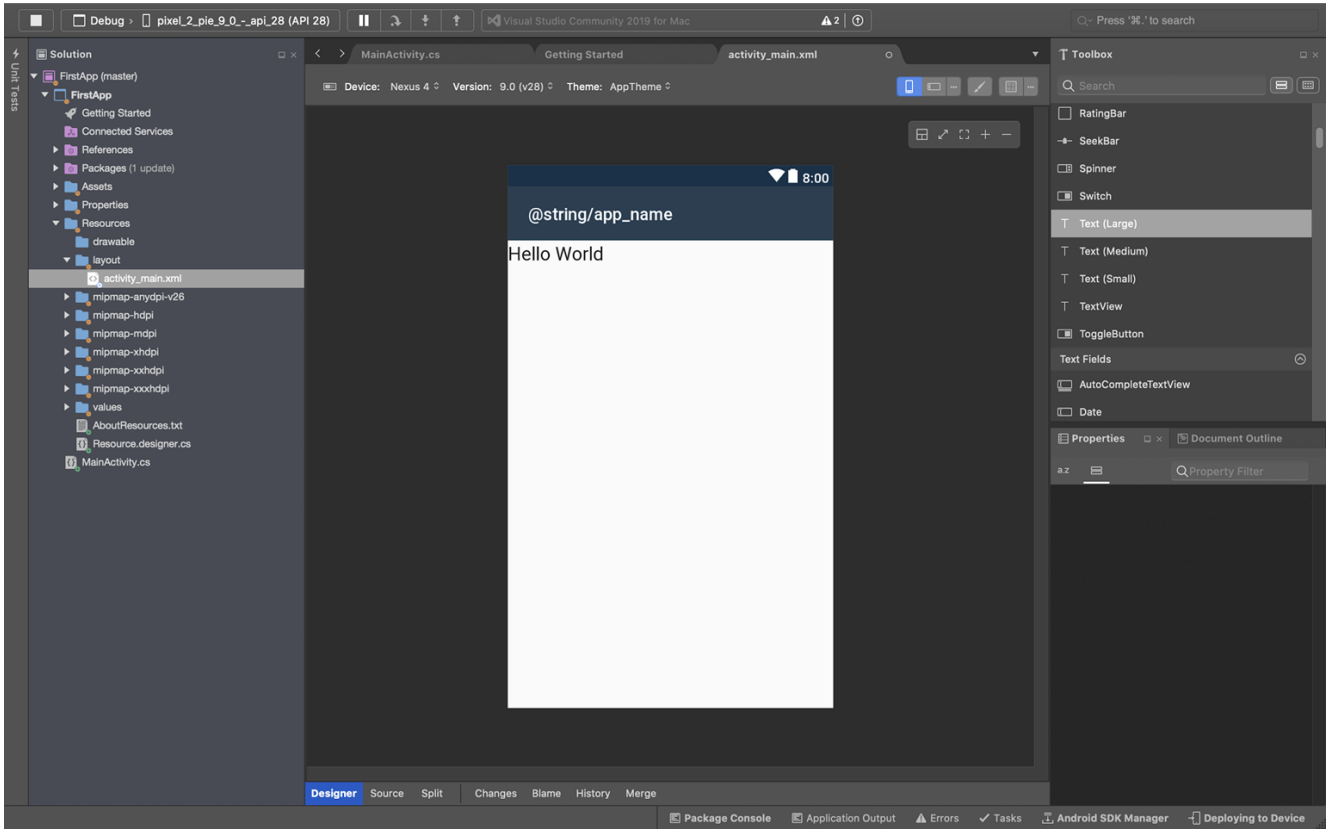
You can create new screens in the future in this directory

The file will be in xml format



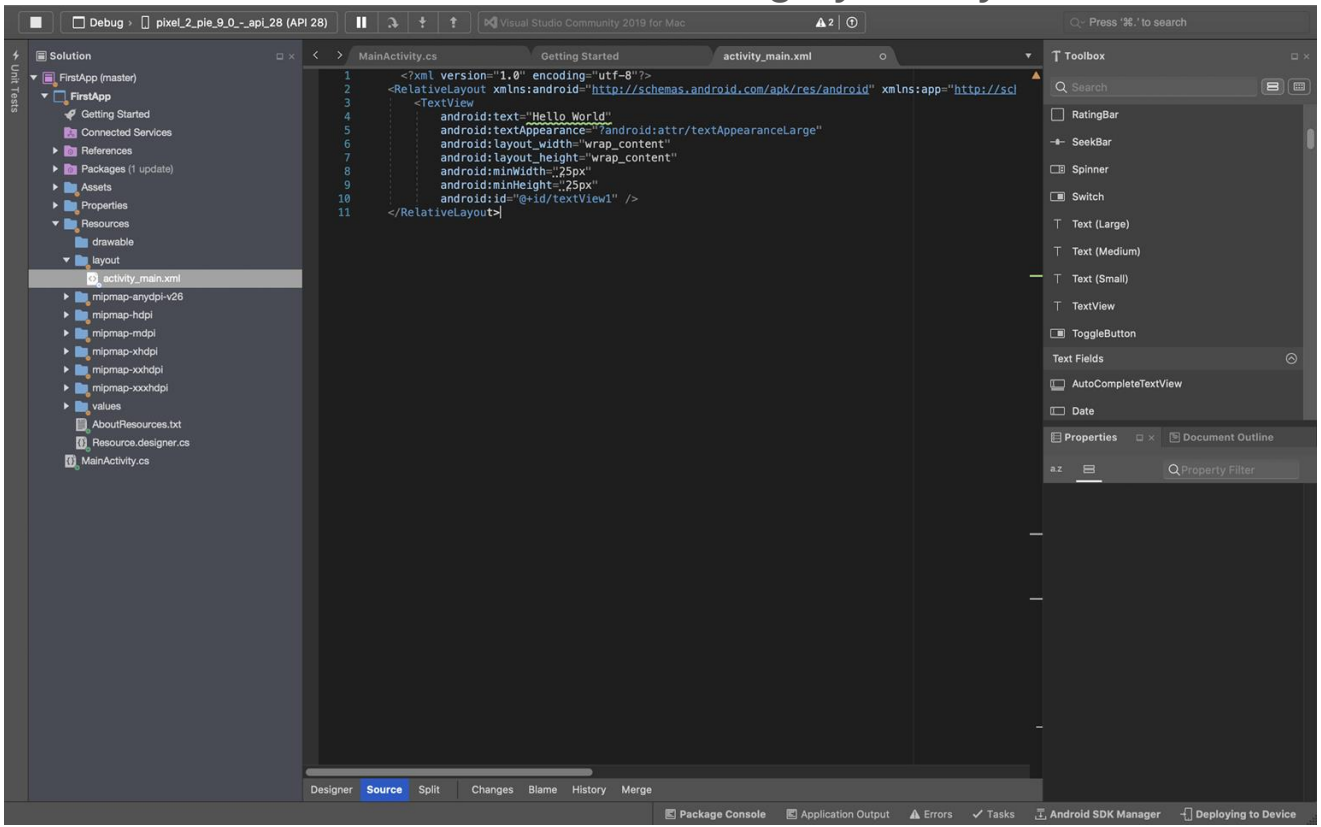
The designer view shows the Graphical User Interface(GUI) of the selected layout

You can drag and drop widgets from the Toolbox on the right into the designer view



The Source view shows the code for the layout

You can edit the source code to further design your layout



Let’s Start Coding!!

Hello World

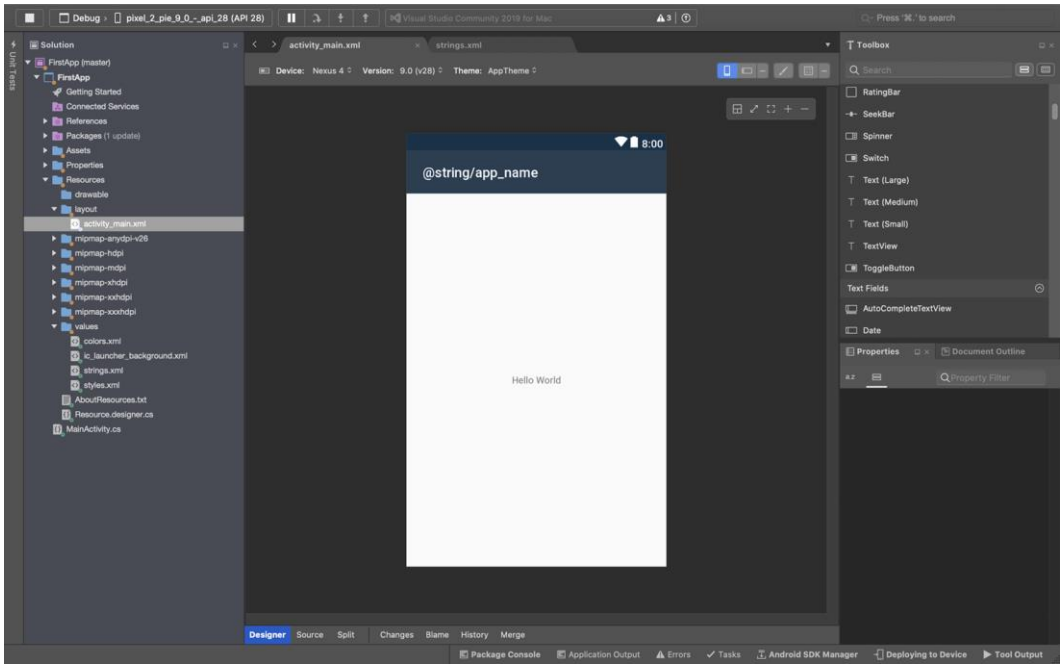
In the activity_main.xml file, type the following code in the RelativeLayout tag:

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:minWidth="25px"
    android:minHeight="25px">

    <TextView
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:id="@+id/lblHelloWorld"
        android:text="Hello World"
        android:gravity="center"/>
</RelativeLayout>
```

Hello World

Go to the Designer view

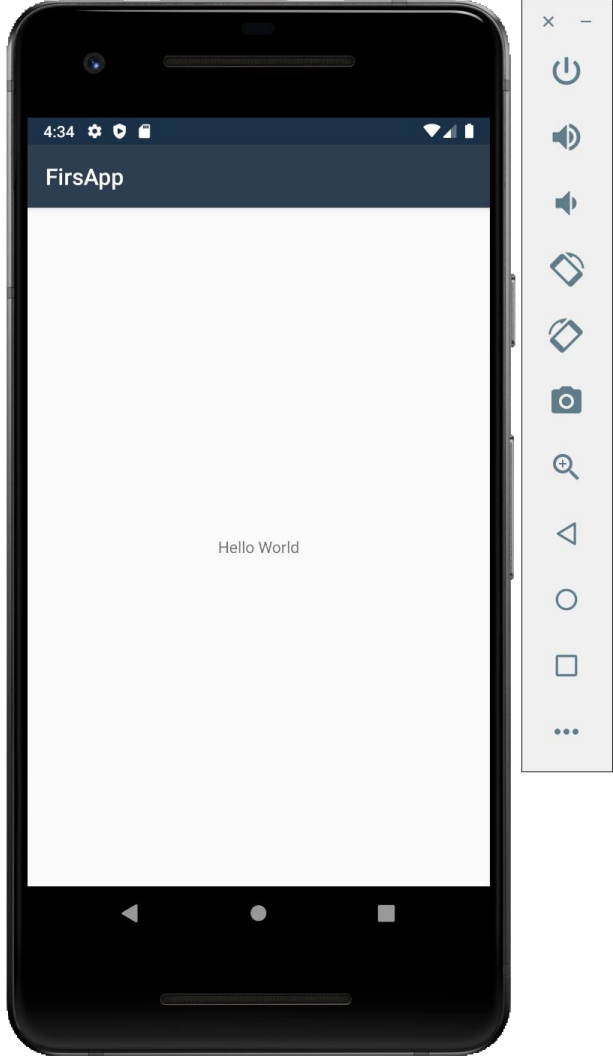
Your screen should look like this



Hello World

Run your project on your emulator

Your screen should look like this



Xaml Essentials

In xaml xamarin go to Mainpage.xaml.

There u will see <ContentPage….> do not remove the content page tags

What are these < > these indicate tags. Each tag has a starting tag like this

<ContentPage> and an end tag </ContentPage>. Every tag must have a start and end.

To end a tag faster, u can do this <ContentPage …… /> this also ends a tag, but allows u to edit some properties.

Label Element

Label Element

On the xaml page, create a Label tag

Now save the xaml file and then run the emulator.

U will see that the Hello is at the top left of the page.

Now remove this and try this way of writing, u get

The same display on the emulator.

```
<?xml version="1.0" encoding="utf-
<ContentPage xmlns="http://xamarir
             xmlns:x="http://schem
             xmlns:d="http://xamar
             xmlns:mc="http://sche
             mc:Ignorable="d"
             x:Class="AppTesting.M
             xmlns:ic="clr-namespa
             xmlns:local="clr-name
             >

    <Label Text="Hello!"/>
```

```
<Label>
    Hello!
</Label>
```

Label Element and StackLayout

```
<Label>
    Hello
</Label>
<Label Text="John"/>
```

Now try to add another Label element with ur name inside

The blue squiggly line appears underneath because we cannot do this. As the 1st label property with the text hello occupies the entire screen. We are unable to add another label element inside.

We solve this problem by using the StackLayout element which allows elements inside it to stack on top of each other.

```
                >
<StackLayout>
    <Label>
        Hello
    </Label>
    <Label Text="John"/>
</StackLayout>
```

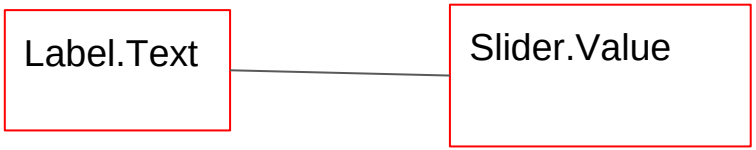
Now run the emulator and see, u can see that Hello

Is stacked on top of John.

Data Binding

Now lets try something else, lets use the Slider element. So now add Slider element below our label elements. Run the emulator to see how slider looks like. Now we would like to try this concept called data binding.

We want to bind the label.text to the slider.value



We do this so that when our slider changes value, our text also changes to the slider value. For example when our slider is halfway, we wan the text to display 0.50.

Data Binding

So to bind it, we need to first add a name for our slider. How do we do this? Within the Label tag, we add x:Name = “slider”. This is how we do naming for xaml.

To make our slider more visible, within the label tag set the BackgroundColor property to blue.

Now for databinding,

In the label tag, set Text=”{Binding Source={x:Reference slider}, Path=Value, StringFormat= ‘Value is 0:{F2}’ }”

Data Binding

So lets break this statement down, { curly braces instructs xaml that this is not a string literal but a xaml mark up extension. Then the word Binding is need as it is a xaml markup extension.

Then we need to specify what object we want to bind the text property to. So we set the Source and then we reference it to slider object. To reference an object we need to use {x:reference slider} which is another markup extension and the name of our slider.

Now we look at which property of the slider we are interested in and we set a path to it. So we set Path=Value. However to make the code neater, we add one more additional thing which is a stringformat. 0 represents first argument and f2 for formatting.

Data Binding

Now run this on the emulator and see the slider.

```
<StackLayout>
    <Label x:Name="label1" Text="{Binding Source={x:Reference slider1}, Path=Value,
        StringFormat='Value is {0:F2}'}"/>
    <Slider x:Name="slider1" BackgroundColor="Blue"/>
</StackLayout>
```

EventHandler

Now remove the entire binding code. We will use an eventhandler to do the same thing. Now we need to name the Label element using x:Name again.

Then within the Slider tag, we use eventhandler called ValueChanged. This can be seen by the lightning symbol next to the ValueChanged which indicates its an event.

After setting valuechange, add new event handler

And see the mainpage.cs code, there u will see a new code has been added.

```
0 references
private void slider1_ValueChanged(object sender, ValueChangedEventArgs e)
{

}
```

EventHandler

Here we add the line label1.Text=String.Format(“Value is {0:F2}”,e.NewValue);

So label1 is the name of our label element, .Text is the property of label1 we are accessing, we set this to a string format.

The string format will display a “value is “ and the {0:F2} is a placeholder that will be replaced by the e.NewValue.

e.NewValue is the parameter in the method that was created for us when we created the eventhandler. As seen it references to ValueChangedEventArgs e in the eventhandler.

Run this on the emulator and u will get the same thing.

Xaml essentials

So u learnt the

Stacklayout

label and slider elements

Data binding

Eventhandler

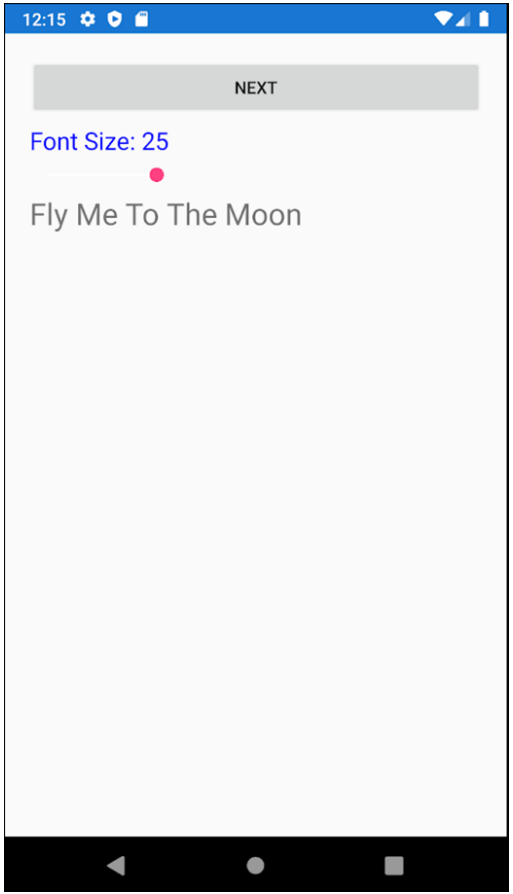
Exercise

Same for all exercises, solutions will be provided but advisable to look at them only after trying to get the same output urself. There is no fixed solution to coding, so yr answer may be better than mine or vice versa.

This exercise is a simple exercise.

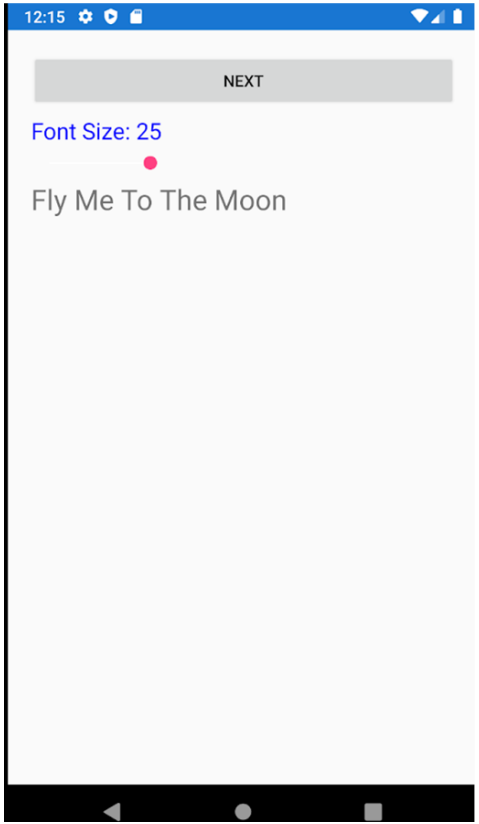
Exercise

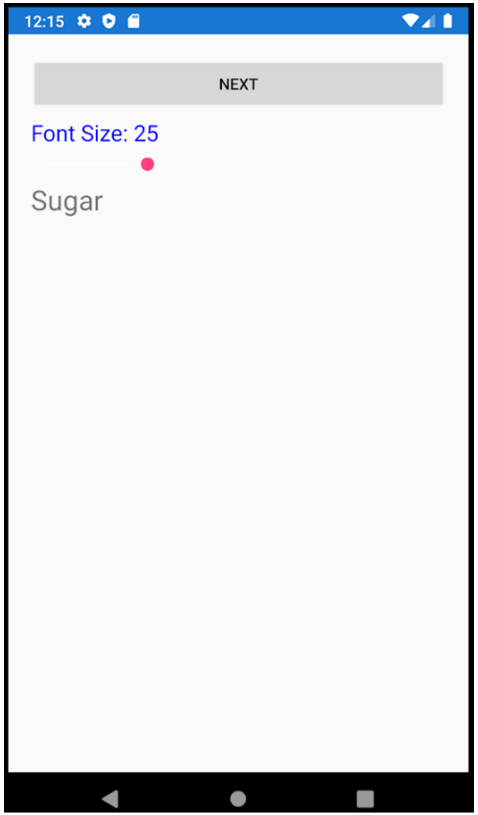
When we move
The slider, we
See that the
Fontsize number
increases
And also the
Fontsize increase
For the song.



Exercise

Upon pressing the Next button, the Name of the song in Grey changes.





Exercise

Task: Build a simple app for browsing through a list of songs.

• your cs page hardcode the songs in a list string[] etc

String[] { …….}

For simplicity, you should just include 4 songs inside.

• Use 25 units padding for the page (25 units on all sides). You can set this directly in the ContentPage element:

Padding=“25”

Exercise

- Set the Maximum and Minimum attributes of slider:

<Slider Maximum=“40” Minimum=“16”/>

- Enable XAML compilation.

Layouts

Stack Layout

Grid

Absolute Layout

Relative Layout

Stack Layout

From the previous section xaml essentials, we looked at the stack layout, in this section we will look into the properties of the stacklayout.

Create a stacklayout with 2 labels and set the text to anything u want.

1. HorizontalOptions and VerticalOptions are very important. Set your stacklayout to HorizontalOptions=”Center” and run the emulator. See how your labels are displayed now. Try the same thing for VerticalOptions=”Center” or both horizontaloptions and verticaloptions at the same time.

Now instead of putting it the Center, try others such as Start/End and run the emulator, see the difference in positioning for this property.

Stack Layout

2. we have BackgroundColor. Remove the verticaloptions and horizontaloptions,

Set backgroundcolor to a color like eg green which will fill the entire stacklayout container with the color u set it as, even the labels inside it with the color. This is because the stacklayout container contains children elements such as labels or buttons, as color fills everything, even its children elements are filled with the color. Try different colors and running the emulator.

Now ensure ur backgroundcolor for stacklayout is set to a color like eg green and now set a diff backgroundcolor for the label to blue. Now try adding in verticaloptions and horizontal options, see how the container for stacklayout shrinks.

Stack Layout

3. Next is the orientation property. This can either be set to vertical or horizontal.

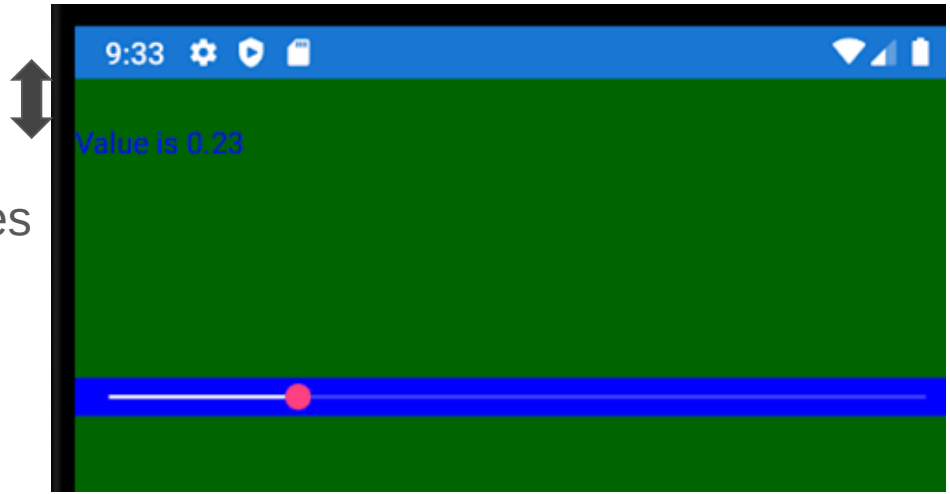
Try each one out and run the emulator

4. Next is spacing property. This means spacing btwn its children elements, for ur case is the spacing btwn 2 labels. Set it to 100 and run the emulator.

5. The Padding property. This is the amount of space between the layout and its children. What do i mean? Set this code in ur visual studios and run the emulator

```
<StackLayout Spacing="100" Orientation="Vertical" BackgroundColor="DarkGreen" Padding="0,20,0,0">
    <Label x:Name="label1" TextColor="Blue"/>
    <Slider x:Name="slider1" BackgroundColor="Blue" ValueChanged="slider1_ValueChanged" />
</StackLayout>
```

Stack Layout



Here the stacklayout container stretches Out to the entire page, this is why the Entire page is green color. Inside the Container, there is a label and a slider Child element.
Padding is the black double sided arrow. It is the spacing of its child elements which is label from the starting of the stacklayout container. But why is this starting only at the top?

Try setting to “20,0,0,0” instead. Then try 0,0,20,0 then 0,0,0,20 and see what each number represents.

Stack Layout

U can even have a stacklayout within a stacklayout

Try it.

Grid Layout

Grid layouts are very useful for calculators, calenders, keypads or photoalbums.

Grid layout is similar to stacklayout but has some differing properties.

It has a different syntax, so here we set 6 labels into rows and columns as shown

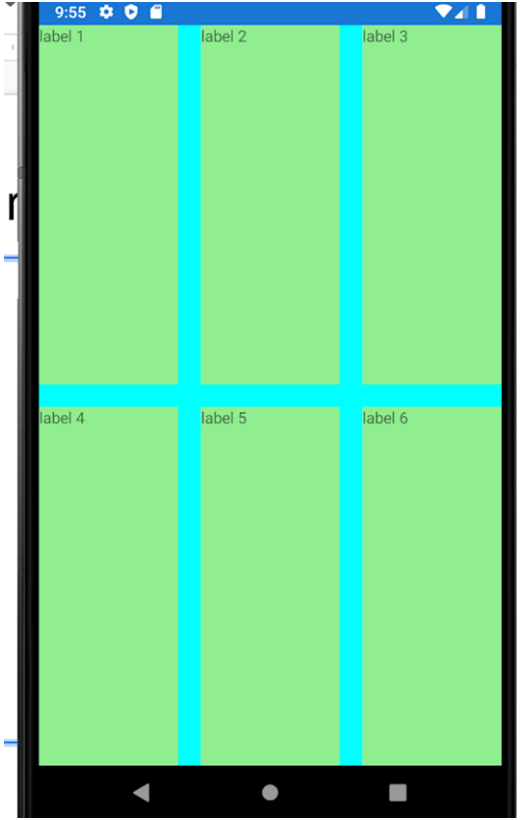
```
<Grid>
    <Label Grid.Row="0" Grid.Column="0" Text="label 1"/>
    <Label Grid.Row="0" Grid.Column="1" Text="label 2"/>
    <Label Grid.Row="0" Grid.Column="2" Text="label 3"/>
    <Label Grid.Row="1" Grid.Column="0" Text="label 4"/>
    <Label Grid.Row="1" Grid.Column="1" Text="label 5"/>
    <Label Grid.Row="1" Grid.Column="2" Text="label 6"/>
</Grid>
```

Run the emulator and see for urself.

The rows and columns are not that defined. Lets change this.

So within the Grid tag, set Backgroundcolor to a color like Aqua, and within the label tag set each label tag to a color like lightgreen.

Grid Layout



However, we will need to add something else to grid,

We will add the property to the grid tag called RowSpacing And ColumnSpacing, set each to 20. Run the emulator and See how the columns and rows are organized.

Rowspacing and columnspacing are self explanatory.

So next is the Grid.ColumnSpan and Grid.Rowspan Properties

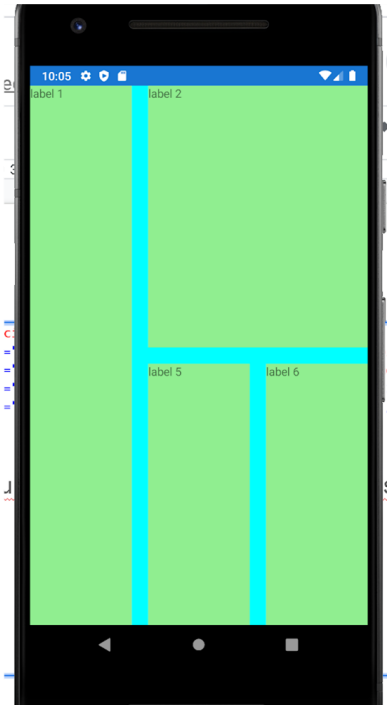
ColumnSpan means u combine columns. So eg ColumnSpan = “2” means we combine 2 columns into 1. Rowspan is the same but combine rows

Grid Layout

```
<Grid BackgroundColor="Aqua" RowSpacing="20" ColumnSpacing="20">
    <Label Grid.Row="0" Grid.Column="0" Text="label 1" Grid.RowSpan="2" BackgroundColor="LightGreen"/>
    <Label Grid.Row="0" Grid.Column="1" Text="label 2" Grid.ColumnSpan="2" BackgroundColor="LightGreen"/>
    <Label Grid.Row="1" Grid.Column="1" Text="label 5" BackgroundColor="LightGreen"/>
    <Label Grid.Row="1" Grid.Column="2" Text="label 6" BackgroundColor="LightGreen" />
</Grid>
</ContentPage>
```

Try this code and see for urself the way rowspan and

column span works



Grid Layout

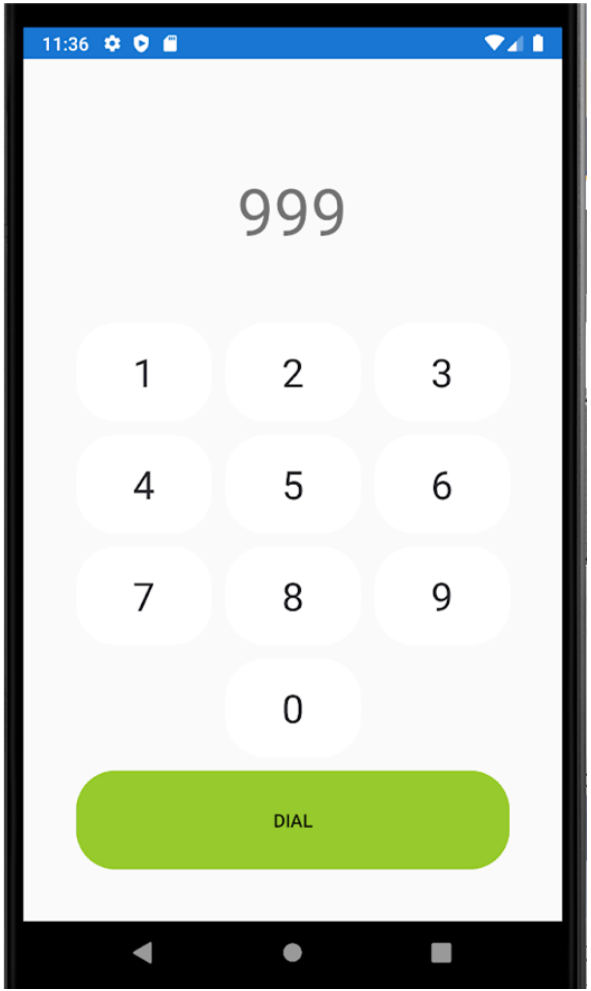
```
<Grid BackgroundColor="Aqua" RowSpacing
    <Grid.RowDefinitions>
        <RowDefinition Height="*"/>
        <RowDefinition Height="*"/>
    </Grid.RowDefinitions>
    <Label Grid.Row="0" Grid.Column="0"
```

Lastly add this element grid.rowdefinitions to ur code and you will be able to set the height. * means that all rows will be proportional, u can set specific units like 100. If you want one row to be twice the height of the other you can set 2*. Play around with this and also grid.columndefinitions to adjust width.

Exercise

In this exercise, we are going to make a phone Interface. Here we do not want to add function for the buttons. (meaning we dont want the Buttons to have a clicked event)
We just want you to make the layout exactly like This.

Hint: Use grid layout.



Exercise

Page background color: #354242

Phone number label
- FontSize = “40”

Numeric buttons
- FontSize = “27”
- TextColor = “#28282c”
- BackgroundColor = “#FFF”
- BorderRadius = “25” (rounded corners)

Exercise

Dial button

- BackgroundColor = “#96ca2d”

Grid

- Padding = “35”
- RowSpacing = “10”
- ColumnSpacing = “10”
- First row is twice taller than other rows

Absolute Layout

The 3rd layout we are gonna learn is the absolutelayout.

Absolute Layout is used to anchor elements to the edge of the screen and to overlay elements.

What does this mean? This is basically used for positioning of the elements inside this layout and we are laying certain elements over other elements.

Absolute Layout

Like the stacklayout and grid, absolutelayout stretches to fill its container when we set a color to it. Likewise, try it by setting backgroundcolor to a color u like.

Next we have 2 attached bindable properties that are the entire crux of absolute layout. These 2 are essential to set the position and size of each element inside the absolute layout.

1. First property is called LayoutBounds
2. Second property is LayoutFlags

Absolute Layout

It is a rectangle that determines the position and size of this element. There are 4 numbers that represent the X , Y , Width and Height.

Numbers passed into these 4 items are either absolute or proportional values. Absolute values can be any number but Proportional values are doubles from 0 to 1.

So what do i mean for x and y axis and width and height exactly? This will be answered in the next slide.

Absolute Layout

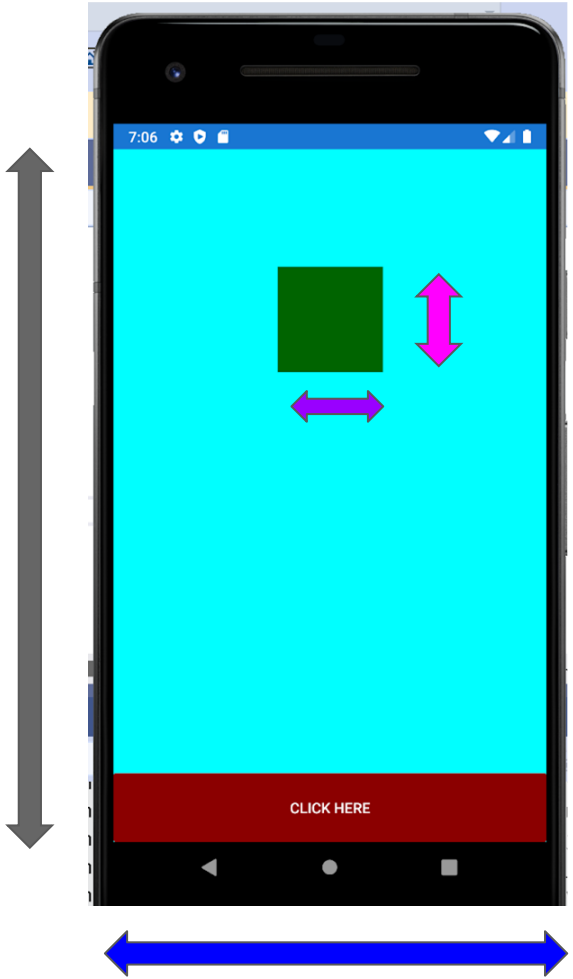
X axis is denoted by the blue arrow

Y axis is denoted by grey arrow

Width is denote by pink arrow

Height is denoted by purple arrow

So x and y axis is basically the entire screen of the Phone. Width and Height are only for the object.



Absolute Layout layout bounds.

So from the answer look at how this is done. For the boxview which is the green box, 0.5, 0.2 are proportional values. 100 and 100 are absolute values.

For the button, 0,1,1,0.1 are all proportional values.

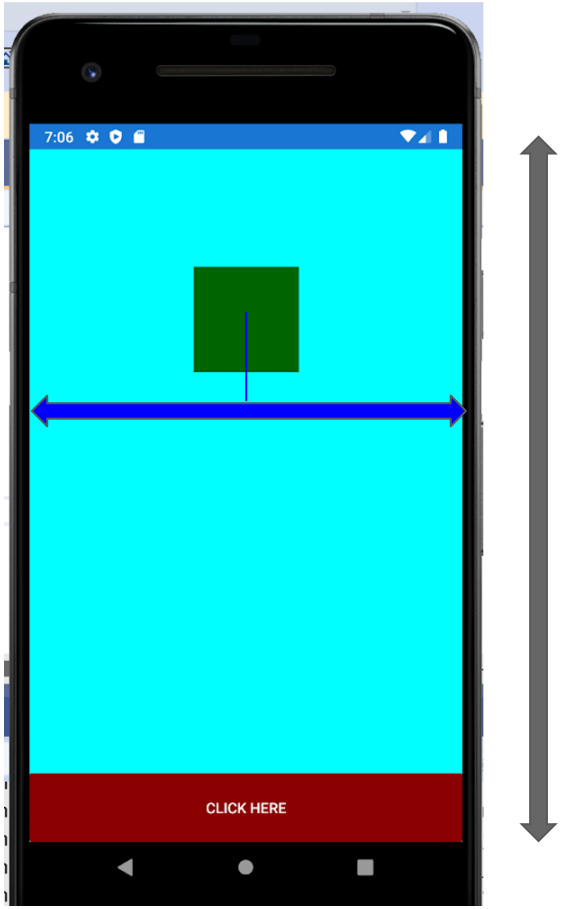
```
<AbsoluteLayout BackgroundColor="Aqua">
    <BoxView BackgroundColor="DarkGreen"
             AbsoluteLayout.LayoutBounds="0.5,0.2,100,100"
             AbsoluteLayout.LayoutFlags="PositionProportional"/>
    <Button AbsoluteLayout.LayoutBounds="0,1,1,0.1"
            AbsoluteLayout.LayoutFlags="All"
            Text="Click Here" BackgroundColor="DarkRed" TextColor="White"/>
</AbsoluteLayout>
```

Absolute Layout

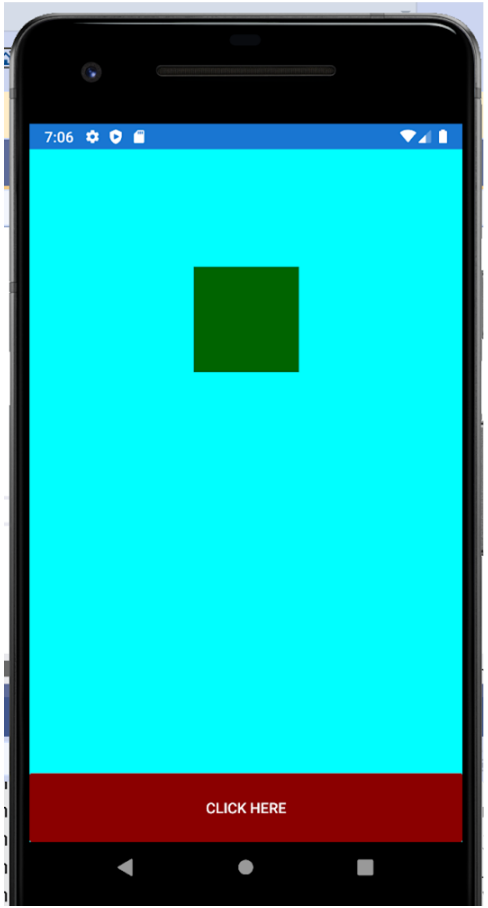
So now following the same coloured arrows.
0.5 of x axis, u can see that the green box is positioned in
The middle of the x axis. Thats why its 0.5

For the y axis, the position is about 0.2 of the y axis

For the width and height they are both 100 units each
And are absolute values.



Absolute Layout



For the button, all values are proportional. As u can see
With regards to the x axis, the value is 0. Doesnt that
Mean it shld be all the way at the left? Yes. But however,
This is not relevant as the width is a proportional value
That is 1. So, no matter what we put for this x value, btwn
0 to 1 it doesnt matter as the width is 1 which occupies
The entire X axis already. Try changing the x for this code.

For the y axis we put 1 as we want it all the way to the
Bottom of the screen. 0 means close to the top, 1 means
Close to the bottom. Same for x axis, 0 to the left, 1 to
The right. Then we want the button to be height 0.1 of the Y axis of the screen.

Absolute Layout Flags

So from the example, there is an additional property which i havent explained yet, here it is.
Here are the values you Can use for the layout Flags.

- `None`, indicates that values will be interpreted as absolute. This is the default value of the AbsoluteLayout.LayoutFlags attached property.
- `XProportional`, indicates that the `x` value will be interpreted as proportional, while treating all other values as absolute.
- `YProportional`, indicates that the `y` value will be interpreted as proportional, while treating all other values as absolute.
- `WidthProportional`, indicates that the `width` value will be interpreted as proportional, while treating all other values as absolute.
- `HeightProportional`, indicates that the `height` value will be interpreted as proportional, while treating all other values as absolute.
- `PositionProportional`, indicates that the `x` and `y` values will be interpreted as proportional, while the size values are interpreted as absolute.
- `SizeProportional`, indicates that the `width` and `height` values will be interpreted as proportional, while the position values are interpreted as absolute.
- `All`, indicates that all values will be interpreted as proportional.

Absolute Layout Flags

So following the previous example, for the boxview i only want the x and y values to be treated as proportional values. I want my width and height to be treated as absolute values thats why i use PositionProportional.

For the button, I use All because all my values are proportional. Play around with the values and see for urself. I personally favour using all proportional values as its easier.

```
<AbsoluteLayout BackgroundColor="Aqua">
    <BoxView BackgroundColor="DarkGreen"
             AbsoluteLayout.LayoutBounds="0.5,0.2,100,100"
             AbsoluteLayout.LayoutFlags="PositionProportional"/>
    <Button AbsoluteLayout.LayoutBounds="0,1,1,0.1"
            AbsoluteLayout.LayoutFlags="All"
            Text="Click Here" BackgroundColor="DarkRed" TextColor="White"/>
</AbsoluteLayout>
```

Absolute Layout

So the layoutbounds and layoutflags work hand in hand. Layout bounds are the values u specify however, u need layoutflags to tell the xaml compiler which values are proportional and which are absolute.

Also do take note for layout flags, if u set something like

LayoutBounds=”0,1,1,50” LayoutFlags=”PositionProportional,WidthProportional” This helps tell ur compiler that your x and y and width are proportional but your height is an absolute value.

Do play around with this u can even create layouts like instagram layouts.

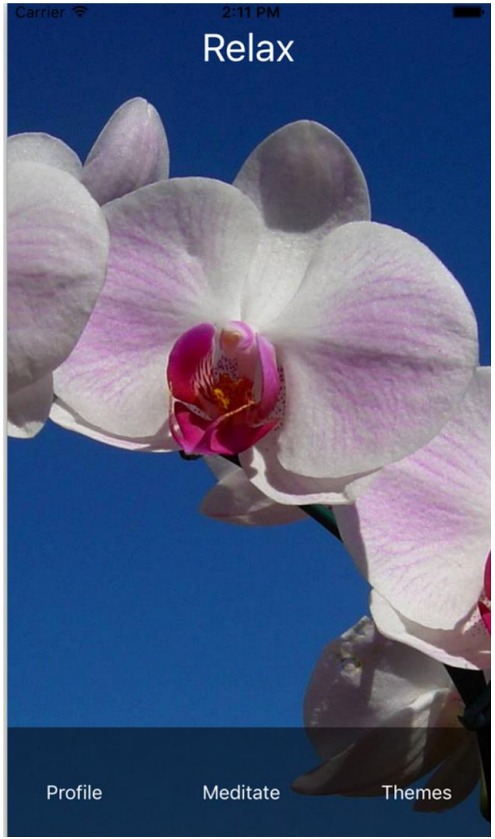
Exercise

Absolute Layout with one label at the top

An image with a background

And 3 labels in a stack layout

on top of a boxview



Exercise

Here we are overlaying a few labels on top of an image. So, that’s why we need to use AbsoluteLayout.

Also, we want to make sure that the 3 labels (Profile, Picture and Theme) are always near the bottom of the page. That’s another reason why we should use AbsoluteLayout. The labels are in a horizontal StackLayout, which is on top of a black BoxView with Opacity of 50%. This is how we get the “bar” effect.

Exercise

Background image

- Source = “http://lorempixel.com/1920/1080/nature/7”
- Aspect = “AspectFill” (dimensions of the image are different from the device.

AspectFill makes the image fill the page irrespective of the size of the device.)

Relax label

- FontSize = “30”
- Should be 20 units from top of the page.

Exercise

BoxView
- anchored to the bottom of the page.
- Its height = 100 units.
- Opacity = “60”.

StackLayout on the bottom
- Padding = “25, 0”.
- anchored to the bottom of the page.
- Its height = 100 units.

Exercise

Labels in the StackLayout

- FontSize = “18”
- HorizontalOptions = “Center”
- For the second label set HorizontalOptions to “CenterAndExpand”. This allocates any available space to this label. To better understand this, set a different background color on each label.

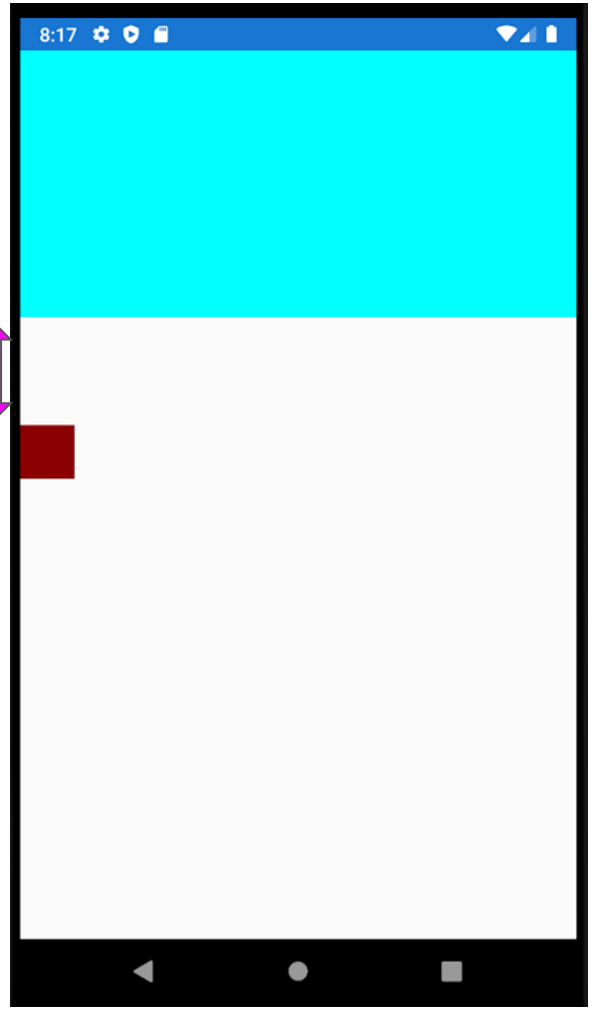
Relative Layout

RelativeLayout is very similar to absoluteLayout. Whatever can be done in absolute layout can also be done for relative layout.

RelativeLayout is used to overlay elements, apply constraints based on another element and for more control over position and size of elements.

Relative Layout

From this example, I would like to set this brown box Such that it is always at a constant of 80 units from the Aqua box at the top. But then wouldnt i use absolute Layout? Yes. but for this case since i wan a constant 80 units as indicated by the pink arrow, I can only Set this using relative layout. AbsoluteLayout u are Unable to set it such that i wan my brown box to be 80 units away from the aqua box.



Relative Layout

Below is the code to do this aqua and brown box.

```
<RelativeLayout>
    <BoxView Color="Aqua" x:Name="banner"

             RelativeLayout.WidthConstraint="{ConstraintExpression
        Type=RelativeToParent, Property=Width, Factor=1}"

             RelativeLayout.HeightConstraint="{ConstraintExpression Type=RelativeToParent,
        Property=Height, Factor=0.3}"/>

    <BoxView Color="DarkRed" RelativeLayout.XConstraint="{ConstraintExpression

        Type=RelativeToParent, Property=Width, Factor=0}"

             RelativeLayout.YConstraint="{ConstraintExpression Type=RelativeToView,
        ElementName=banner, Property=Height, Factor=1, Constant=80}"/>
</RelativeLayout>
```

Relative Layout

The main property here is eg RelativeLayout.XConstraint. However this has a few more options, YConstraint, WidthConstraint, HeightConstraint and finally BoundsConstraint.

- XConstraint, of type Constraint, which is an attached property that represents the constraint on the X position of the child.
- YConstraint, of type Constraint, which is an attached property that represents the constraint on the Y position of the child.
- WidthConstraint, of type Constraint, which is an attached property that represents the constraint on the width of the child.
- HeightConstraint, of type Constraint, which is an attached property that represents the constraint on the height of the child.
- BoundsConstraint, of type BoundsConstraint, which is an attached property that represents the constraint on the position and size of the child. This property can't be easily consumed from XAML.

Relative Layout

So just like absolute layout the 4 numbers, the x constraint will set the position of the element in relation to the x axis.

The y constraint will set the position of the child element in relation to the Y axis.

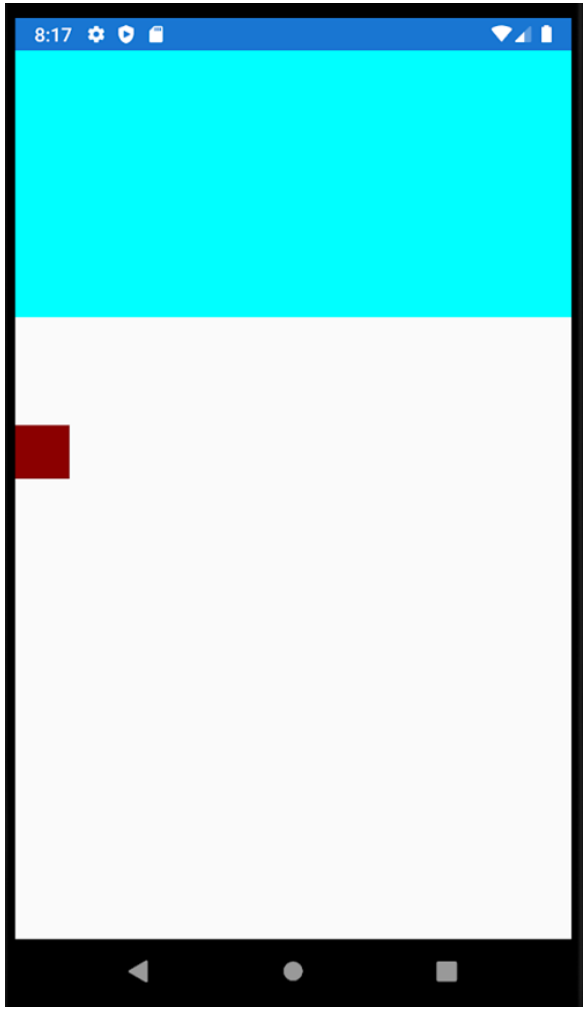
The width constraint will set the width of the element like how wide the element will be.

The height constraint will set the height of the element like how tall the element will be.

Relative Layout

Back to the example code, we now look at the code For RelativeLayout=WidthConstraint first. So here we Set a markup language to bindingconstraint. We then Set our reference for the type to relativetoparent in this Case the parent is the RelativeLayoutContainer that Occupies the entire screen.

Then our property we wan to target is the width of the Child element. Then we set the factor which means How much to multiply by in relation to the parent. So Here we set 1, means our width is the parents width Times 1. So basically same width as the parents.



```
RelativeLayout.HeightConstraint="{ConstraintExpression Type=RelativeToParent,
Property=Height, Factor=0.3}"/>
```

Relative Layout

Next is the HeightConstraint. So its mostly similar to width constraint except the property and the factor. In this case we dont wan to cover the entire screen so we just reference how tall our object is to the screen which is 0.3. So our element will be 0.3 the height of our parent.

Then for the next boxview the brown one, do note we use x and y constraint and not height and width constraint. Why do we use this? Because x and y constraint controls the position of the brown box. Unless u wan the brown box to be bigger than u use height and width constraint.

So here, for the brown box Y constraint is the axis we want to look at here. As our objective is to set the brown box 80 units from the aqua box.

```
RelativeLayout.YConstraint="{ConstraintExpression Type=RelativeToView,
ElementName=banner, Property=Height, Factor=1, Constant=80}"/>
iveLayout>
```

Relative Layout

So looking at the code for the brown box, we see that here we have 2 additional properties. One is that our type is changed to relativetoview instead of relativetoparent. What this means is we want to reference to another element instead of relative layout. But what do we want to reference to?

Thats where our ElementName property comes in. we want to reference it to banner. Banner is the name for our aqua box that we need to set becoz we need to know specifically which boxview to refer to as there could be many.

Now the last additional property is constant. This is where we get the additional 80 units. So together with the factor property, this will xaml that we want the y coordinate value of the brown box to be the height of the banner + 80 units.

Relative Layout

The takeaway here is to know the differences btwn the XConstraint vs WidthConstraint and YConstraint vs HeightConstraint.

Width/Height is to control size of ur element

X and Y is to control the position of ur element.

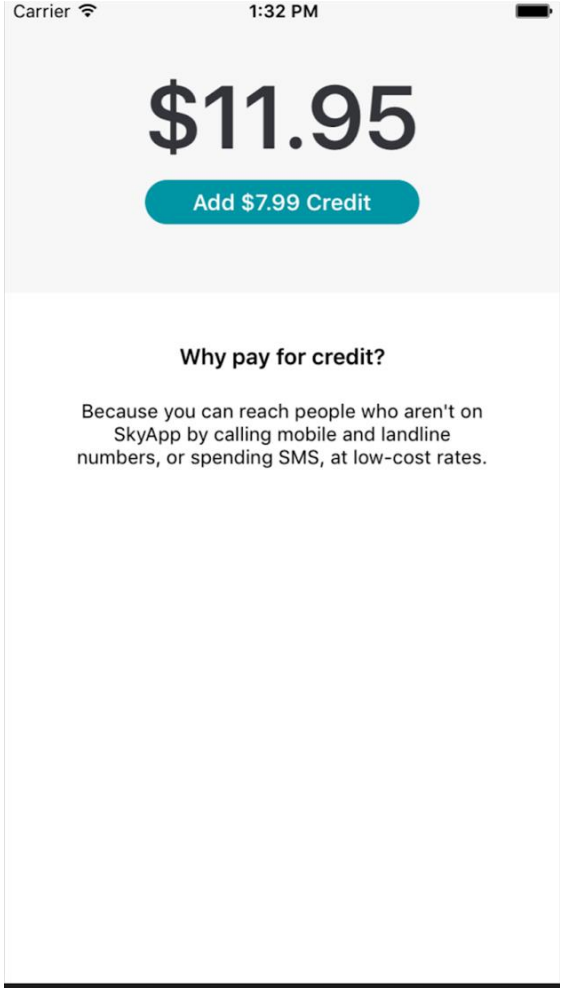
We are done with Layouts!

Exercise

Here we have quite a few items. But think about Why do we need to use relative layout instead of Stacklayout.

For this case we could actually use either one.

But here lets say we would like to have the description to be a constant number of units away from the light grey banner. Thats why we need to use Relative layout.



Exercise

we set the height of the banner to about a third (0.3) of the page height, and the y coordinate of the StackLayout that contains the description to half of the page height.

So, the space between the banner and the description is 0.2 of the page height. The bigger the device, the more space between these two elements.

Sometimes this may be perfectly fine, but other times you may want to keep this space fixed.

So, let’s imagine we want the description to be always 30 units below the banner. Now, it’s better to implement this UI using RelativeLayout.

Exercise

Banner
- BackgroundColor = “#f7f7f7”
- Should take 1/3 of the page height.

$11.95 label
- TextColor = “#33353a”
- FontSize = “60”
- FontAttributes = “Bold”

Exercise

Button

- BackgroundColor = “#1695A3”
- BorderRadius = “15”
- HeightRequest = “30”
- FontSize = “15”
- FontAttributes = “Bold”

StackLayout that contains the description

- Should be 30 units below the parent.

Exercise

“Why pay for credit” label

- FontSize = “15”
- FontAttributes = “Bold”
- Description label • FontSize = “13”

Final Challenge

- Calculator app with all its functionality
- Assume user only calculates 1 equation at a time – meaning 8 + 8 then 64 – 3, and not 8 + 8 – 3
- So + function needs to add 2 numbers together
- - function will take 1st number minus 2nd number
- * function will be 1st number multiply by 2nd number
- / function will be 1st number divide by 2nd number
- AC will reset calculator to 0
- DEL will delete number/math symbol. Eg 1+1 when you press DEL, it will become 1+, also if you have no numbers entered and you press DEL, it will remain at 0

Final Challenge

- = should be the function doing the calculations
- Hint: User grid layout, use buttons, label