

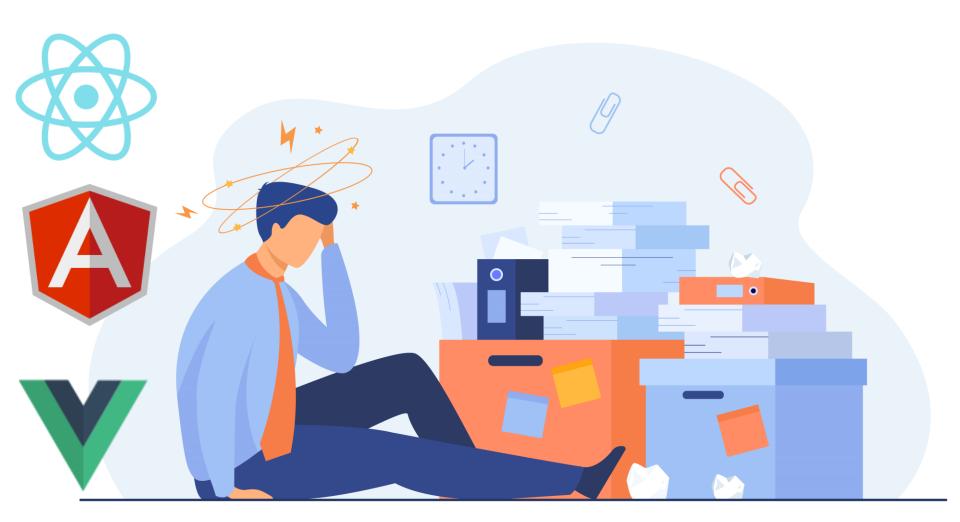


DOTNETMASTERY

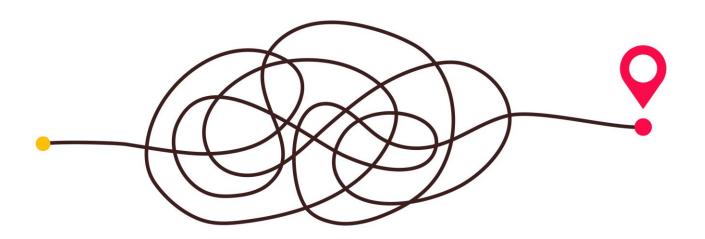
The ultimate learning recourse for .NET developers.

• What is this course about?

Why this course?



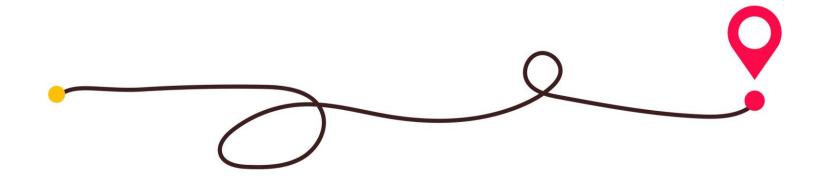
My learning curve!



NOT! Your Journey



Your Journey!

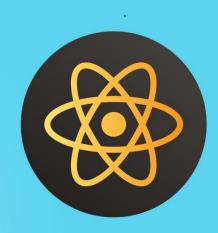




Best Practices JOB **JOB JOB**







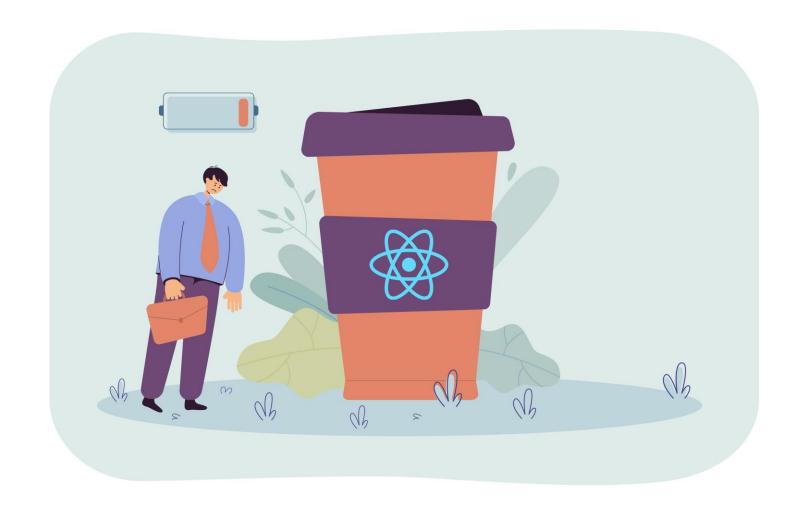






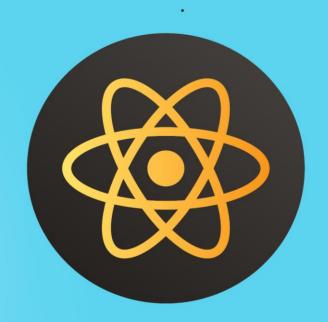


25 Assignments











DOTNETMASTERY

The ultimate learning recourse for .NET developers.

What is React?

What is React?

Create React App

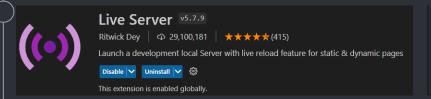
• What is React?

- React is Front End Javascript Framework
- React is Declarative
- React is Composable / Component based
- React is FAST!
- Maintained by META
- React is responsible for maintaining/managing the UI Aspect.

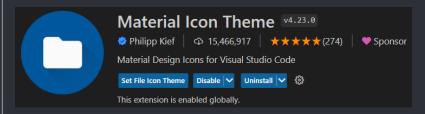
Tools Needed

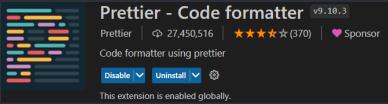
- Visual Studio Code
- Node JS
- Visual Studio 2022
- SQL Server Management Studio
- Azure Subscription

Visual Studio Code Extensions











ES7+ React/Redux/React-Native snippets V4.4.3

Extensions for React, React-Native and Redux in JS/TS with ES7+ syntax. Customizable. Built-in integration with prettier.







How to get Help?

Start json-server

npx json-server --watch data/traveldb.json --port 5001

Class vs Functional Components

Class Components

- Less common
- render() method is required, which returns an HTML element
- Stateful components
- Class components have lifecycle methods

Functional Components

- More common
- render() method is not present, it directly returns HTML element or nothing.
- Stateless components
- Class components do have lifecycle methods
- Hooks!

- With React version 16.8, class components have taken a back seat to functional components. Functional components are more concise, leading to cleaner, less complex code. They don't include lifecycle methods or inherited members that may or may not be required for code functionality.
- Anything that can be done using class components can also be done using functional components. The only exception is that React supports a special class component called **Error Boundaries** that can't be duplicated as a function component.
 - Because class components extend from React.Component, they have state and lifecycle methods associated with them. Their presence requires you to better understand when lifecycle events will occur and how to respond to them to manage state. Classes also require additional setup for the component or to make API calls for data, primarily implemented via the constructor. Sharing logic between multiple class objects without implementing design patterns is more difficult, leading to code that is more complex and difficult to maintain.
 - Deciding which component to use always devolves into a discussion of <u>inheritance vs composition</u>. Using functional components encourages composition, while class components lend themselves to inheritance design patterns. Currently, composition is considered a best practice in programming, which is why most new React code uses functional components instead of class components. That said, React still supports class components for legacy purposes.

0

JSX Components

Component Names must be Pascal Casing
 function Footer() { <Footer />

Props must be camel casing
 <input maxLength={5} readOnly="false">

Inline style is an object so it must be enclosed in double curly brackets
 Happy Coding!

Integer in props will go in {} and "" will be string, and bool goes in {}
 <input maxLength={5} readOnly={false} placeholder="Ben"></input>

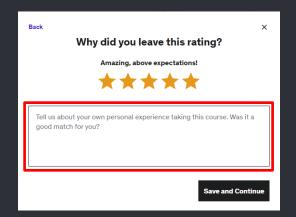
• 'class' needs to be replaced with 'className' <h1 className="heading1">

RTK Query

- Queries used to fetch data (Get/GetALL)
- Mutation used to update data (Create/Update/Delete)

Reviews

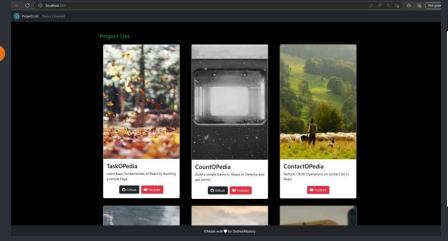


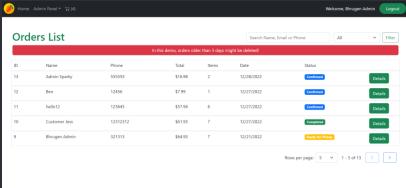


dotnetmastery@gmail.com

Feedback

Bonus Content





My code doesn't work!



- Google it!
- Check Q&A Section
- POST a queston (with Git branch)

Code works, but I am confused



- Google it
- Message me on udemy!

My code doesn't work!



- Google it!
- Check Q&A Section
- POST a queston (with Git branch)

Code works, but I am confused



- Google it!
- · Message me on udemy!

Prerequisites

- Basic programming experience
- HTML / CSS / JS Familiarity
- .NET API CRUD Operations (.NET Core & EF Core

• Reviews

Azure Deployment

Project Resources

• JSX

Projects Built

- 1. TaskOPedia -
- 2. CountOPedia
- 3. ContactOPedia
- 4. LifeCyclOPedia
- 5. WatchOPedia
- 6. RouteOPedia
- 7. ReduxOPedia
- 8. TravelOPedia
- 9. Red Mango

What we will learn

- React Fundamentals
- 2. TypeScript
- 3. Redux Toolkit & RTK Query
- 4. Routing in Redux
- 5. Axios
- 6. Stripe Payment in React
- 7. Authentication and Authorization in React
- 8. File Management in React



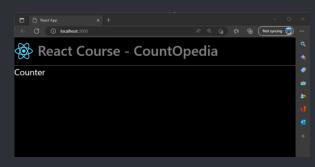
Hello! I AM JAYDEN SMITH

I am here because I love to give presentations. You can find me at @username

Assignment 7 - CountOPedia Setup

- 1. Remove all files other than index.html, index.js and react Image
- 2. Move react image to src/images folder. (Create images folder)
- Clean index.js and index.html (Remove anything that is not used)
- 4. Add Bootstrap js/css CDN inside index.html
- 5. Create 2 components (Design to look as shown in image)
 - Header.jsx (Functional Component)
 - Counter.jsx (Class Component)
- 6. Copy the images provided in course snippet section 4 to the images folder





Class Component Lifecycle Methods

1. ComponentDidMount

Runs after the component has been rendered to DOM



2. ComponentDidUpdate

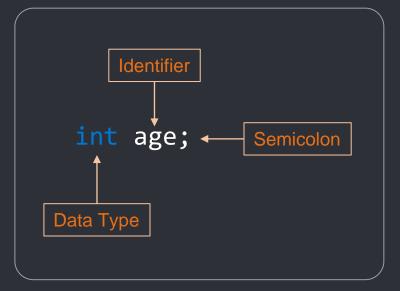
- Runs after a component is updated.
- It is not executed when component is rendered for the first time.

3. ComponentWillUnmount

- Runs after a component is unmounted
- Ideally used for cleanup operations

DataTypes

Declare Variable



Declare & Assign Variable

Assign value of 20 to age

int Age = 20;

Data Types

Byte

- 8-bit unsigned integer
- 0 to 255 (whole numbers)

byte num=100;

Integer

- 32-bit unsigned integer (whole number)
- · -2,147,483,648 to 2,147,483,647

int num=100000000;

Short

- 16-bit unsigned integer
- -32,767 to 32,767 (whole numbers)

short num=10000;

Long

- 64-bit unsigned integer (whole number)
- -9,223,372,036,854,775,808
 +0
 9,223,372,036,854,775,807

Data Types

Float

- ±3.4 x 10³⁸
- \sim 6-9 digits precision

Float num=100.01f;

Decimal

- ±7.9228 x 10²⁸
- 28-29 digits precision

decimal num=1.00920382;

Double

- ±1.7 × 10³⁰⁸
- $\sim 15-17$ digits precision

double num=1.999980099;

Max Decimal : 79228162514264337593543950335 Min Decimal : -79228162514264337593543950335

Max Float : 3.4028235E+38 Min Float : -3.4028235E+38

Max Double : 1.7976931348623157E+308 Min Double : -1.7976931348623157E+308

Result of 1/3:

float: 0.3333333

double: 0.33333333333333333

Float vs Double vs Decimal

- Float 32 bit (7 digits) It is used mostly in graphic libraries because very high demands for processing powers, also used situations that can tolerate rounding errors.
- Double 64 bit (15-16 digits) Double Types are probably the most commonly used data type for real values. Use double for non-integer math where the most precise answer isn't necessary.
- Decimal 128 bit (28-29 significant digits) It gives you a high level of accuracy because of which
 it is used in financial application.

Data Types

Character

- 16-bit
- Single character literal or Unicode.

char temp'A';

String

Multiple characters and numbers

```
string temp="Hello";
```

DataType Range

Туре	Description	Range	Byte
byte	8-bit unsigned integer	0 to 255	1
short	16-bit signed integer	-32,768 to 32,767	2
int	32-bit signed integer	-2,147,483,648	4
		to	
		2,147,483,647	
long	64-bit signed integer	-9,223,372,036,854,775,808	8
		to	
		9,223,372,036,854,775,807	
float	32-bit Single-precision floating point type	±3.4 x 10 ³⁸	4
		~ 6-9 digits precision	
double	64-bit double-precision floating point	±1.7 × 10 ³⁰⁸	8
	15-digit precision	~ 15-17 digits precision	
decimal	128-bit decimal type	±7.9228 x 10 ²⁸	16
		28-29 digits precision	
char	16-bit single Unicode character	Any valid character	2
bool	8-bit logical true/false value	True or False	1

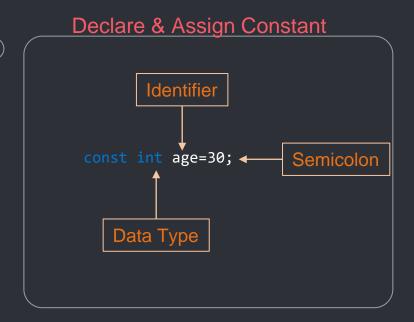


Constants

Constants are immutable values which are known at compile time and do not change for the life of the program

```
const <data type> <constant name> = value;
```

Constants



Naming Convention

- Give your variables meaningful names!
- The name can contain letters, digits, and the underscore character (_)
- The first character of the name must be a letter. The underscore is also a legal first character, but its use is not recommended at the beginning of a name.
- Case matters (that is, upper- and lowercase letters). C# is case-sensitive;
 thus, the names "age" and "Age" refer to two different variables.
- C# keywords can't be used as variable names.
- Camel Casing is the recommended approach for naming variables.
 e.g. thisIsTheVariableName, lowerCase

Type Conversion

Implicit type conversion

No special syntax is required because the conversion always succeeds, and no data will be lost

Explicit type conversion

Explicit conversions require a cast expression.

There is change for data loss!

Implicit Type Conversion

```
int num1 = 100;
```

long Ing1=num1;

Explicit Type Conversion

```
double dou1 = 100;
```

Explicit Type Conversion

```
double dou1 = 100;
int num1 = (int)dou1;  // num 1 = 100
```

Explicit Type Conversion – Data Loss!

```
double doub1 = 900800700600500400;
```

```
// there will be data loss here int intNum = (int) doub1;
```

Explicit Type Conversion (For Non-compatible Types)

```
string age = "10";
int ageValue = (int)age; // gives error
```

Explicit Type Conversion (For Non-compatible Types)

```
int ageValue = Convert.ToInt32(age);
int ageVal = int.Parse(age);
```

Explicit Type Conversion (For Non-compatible Types)

```
Convert.ToInt16();
Convert.ToInt32();
Convert.ToInt64();
Convert.ToDouble();
Convert.ToFloat();
Convert.ToChar();
Convert.ToBoolean();
```

C# Operators

- Assignment Operator
- Arithmetic Operators
- Comparison Operators
- Logical Operators

Assignment Operator

) Basic assignment operator (=) is used to assign values to variables

```
int num;
num = 100;
int num1 = 1000;
```

Arithmetic Operator

Arithmetic operators are used to perform arithmetic operations such as addition, subtraction, multiplication, division, etc.

```
int num1 = 10;
int num2 = 20;
int num3 = num1 + num2 // num3 = 30
```

C# Operators

Arithmetic Operator

Operator	Operator Name	Example
+	Addition Operator	20 + 10 evaluates to 30
-	Subtraction Operator	20 - 10 evaluates to 10
*	Multiplication Operator	20 * 10 evaluates to 200
1	Division Operator	20 / 10 evaluates to 2
%	Modulo Operator (Remainder)	17 % 5 evaluates to 2

Arithmetic Operator (Increment/Decrement)

Operator	Operator Name	Example
++	Increment Operator	x++ (x=x+1)
	Increment Operator	x (x=x-1)

Arithmetic Operators

Postfix Increment

- Increments value by 1
- ++ (X++)
- The value returned by x++ is the value of (x) before the increment takes place

$$x = 11, y = 10$$

Postfix Decrement

- Decrements value by 1
- -- (X--)
- The value returned by x-- is the value of (x) before the decrement takes place

Prefix Increment

- Increments value by 1
- ++ (++X)
- The value returned by ++x is the value of (x)
 after the increment takes place

```
int x = 10;
int y = ++x;
```

Prefix Decrement

- Decrements value by 1
- -- (--x)
- The value returned by --x is the value of (x)
 after the decrement takes place

```
int x = 10;
int y = --x;
x = 9, y = 9
```

Assignment Operator

Operator	Operator Name	Example	
=	Assignment	x=100	
++	Increment Operator	X++	x=x+1
	Decrement Operator	X	x=x-1
+=	Addition Assignment	x+=10	x=x+10
-=	Subtraction Assignment	x-=10	x=x-10
=	Multiplication Assignment	x/10	x=x*10
/=	Division Assignment	x/=10	x=x/10

Comparision Operator

Operator	Operator Name	Example
==	Equal	x==y
!=	Not Equal	x!=y
>	Greater than	x>y
<	Less than	x <y< th=""></y<>
>=	Greater than or equal to	x>=y
<=	Less than or equal to	x<=y

Comparision Operator

X	Υ	Operation	Result
10	10	x==y	True
10	10	x!=y	False
10	10	x>y	False
10	10	x <y< td=""><td>False</td></y<>	False
10	10	x>=y	True
10	10	x<=y	True

Х	Y	Operation	Result
5	10	x==y	False
5	10	x!=y	True
5	10	x>y	False
5	10	x <y< td=""><td>True</td></y<>	True
5	10	x>=y	False
5	10	x<=y	True

Logical Operator

Operator	Operator Name	Example
&&	And	x && y
Ш	Or	x y
!	Not	!x

Logical Operator

Operator	Example	x=3, y=4	x=10,y=1	x=1,y=10
&&	x>5 && y<5	False && True False	True && True True	False && False False
II	x>5 y<5	False True True	True True True	False False False
ı.	!(x>5)	!(False) True	!(True) False	!(False) True

AND and OR Operators

X	у	x&&y	x y	!x
true	true	true	true	false
true	false	false	true	false
False	true	false	true	true
False	false	false	false	true



Strings

A *string* is a series of characters that is used to represent text.

• String's in C#

```
string firstName;
firstName = "Ben";
string lastName = "Spark";
string fullName = firstName + lastName;
string greetingMessage = "Welcome" + fullName;
```

String's in C#

```
string firstName = "Ben";
                                                 Output: Welcome, Ben Spark.
string lastName = "Spark";
string greetingMessage = "Welcome, " + firstName + lastName + ".";
string interpolMessage = $"Welcome, {firstName} {lastName}.";
string formatMessage = string.Format("Welcome, {0} {1}.", firstName, lastName);
```



String's Methods in C#

Functions/methods of the *String* class are used to manipulate and perform different actions on a given string.

Length

```
//index     0123456

//length     1234567
string greeting = "Welcome";

//Gets the length of the string (It is not a method!)
int greetingLength = greeting.Length; // Output -> 7
```

IndexOf

```
string greeting = "Welcome, Ben Spark";
//Get the index of ' ' (space)
int indexOfSpace = greeting.IndexOf(' ');
int indexOfQue = greeting.IndexOf('?');
```

IndexOf returns the first index of the string/char.

```
// Output -> 9
greeting.IndexOf('B');
```

LastIndexOf

```
string greeting = "Welcome, Ben Spark";
//Get the last index of ' ' (space)
int lastIndexOfSpace = greeting.LastIndexOf(' ');
                                                     IndexOf returns the first index of the
                                                               string/char.
int lastIndexOfQue = greeting.LastIndexOf('?');
```

IndexOf

```
string greeting = "Welcome, Ben Spark";
int indexOfBen = greeting.IndexOf("Ben");
int indexOfBill = greeting.IndexOf("Bill");
```

IndexOf returns the first index of the string/char.

LastIndexOf

```
string greeting = "Welcome, Ben Spark Ben";
int indexOfBen = greeting.LastIndexOf("Ben");
int indexOfBill = greeting.LastIndexOf("Bill");
```

SubString

```
String index starts from 0
string greeting = "Welcome, Ben Spark";
string fullName = greeting.Substring(9);
                                                      greeting.Substring(9,3) -> "Ben"
string firstFive = greeting.Substring(0,5);
```

ToUpper / ToLower

```
string firstName = "Ben";
string lastName = "Spark";
string greetingMessageUpper = "Welcome, " + firstName.ToUpper() + ".";
string greetingMessageLower = "Welcome, " + firstName.ToLower() + ".";
```

ToUpper / ToLower

```
string firstName = "Ben";
string lastName = "Spark";
string greetingMessageUpper = "Welcome, " + firstName.ToUpper() + ".";
string greetingMessageLower = "Welcome, " + firstName.ToLower() + ".";
```

Trim

```
string greeting = " Welcome
greeting.Trim(); // Output -> "Welcome"
greeting.TrimStart(); // Output -> "Welcome"
greeting.TrimEnd(); // Output -> " Welcome"
```

IsNullOrEmpty / IsNullOrWhiteSpace

IsNullOrEmpty - checks if a string is null or empty (contains no data).

```
IsNullOrWhiteSpace - checks if a string is null or empty or if a string has only whitespaces.
string greeting = "Welcome";
string greetingEmpty = "";
string greetingWhiteSpace = " ";
string greetingNull = null;
Console.WriteLine(string.IsNullOrEmpty(greeting)); // Output -> False
Console.WriteLine(string.IsNullOrEmpty(greetingEmpty)); // Output -> True
Console.WriteLine(string.IsNullOrEmpty(greetingWhiteSpace)); // Output -> False
Console.WriteLine(string.IsNullOrEmpty(greetingNull)); // Output -> True
Console.WriteLine(string.IsNullOrWhiteSpace(greeting)); // Output -> False
Console.WriteLine(string.IsNullOrWhiteSpace(greetingEmpty)); // Output -> True
Console.WriteLine(string.IsNullOrWhiteSpace(greetingWhiteSpace)); // Output -> True
Console.WriteLine(string.IsNullOrWhiteSpace(greetingNull)); // Output -> True
```

Compare Strings

```
string firstName = "Bhrugen";
string lastName = "";

Console.WriteLine(firstName == lastName); // Output -> False
Console.WriteLine(firstName.Equals(lastName)); // Output -> False

lastName = "Bhrugen";
Console.WriteLine(firstName == lastName); // Output -> True
Console.WriteLine(firstName.Equals(lastName)); // Output -> True
```

String and Int Conversion

```
string age = "20";
string ageTemp = "Test";
int balance = (int)age;
int age1 = Convert.ToInt32(age);
int age2 = int.Parse(age);
int age3 = int.Parse(ageTemp); //Throws Error
int age4;
int.TryParse(ageTemp,out age4);  //Does not throw error, returns 0
```

String and Int Conversion

```
string num = "20000";

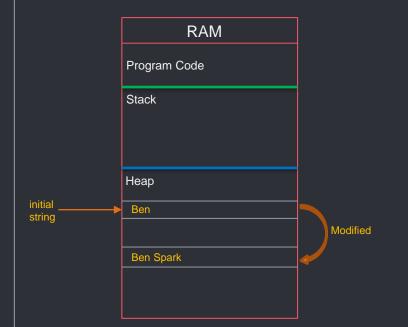
string tempString1 = num.ToString();  // "20000"

string tempString2 = num.ToString("C");  // "$20,000.00"
```

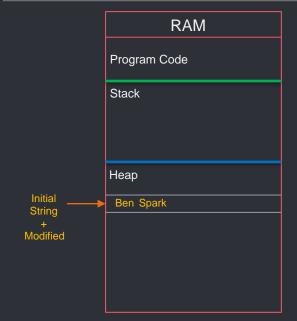
C# StringBuilder

String vs StringBuilder

```
string name = "Ben";  // "Ben"
name = name + " Spark";  // "Ben Spark"
//Strings are immutable
```



```
StringBuilder builder = new StringBuilder();
builder.Append("Ben");  // "Ben"
builder.Append(" Spark");  // "Ben Spark"
//StringBuilder are mutable
```



StringBuilder

- Defined in System.Text
- StringBuilder is Mutable.
- Recommended if you are modifying string too many times.
- StringBuilder does not contain methods available in string like
 - IndexOf()
 - Contains()
 - LastIndexOf(
- New methods in StringBuilder for manipulation
 - Append()
 - Remove()
 - Clear()



Method / Function

A *method/function* is a block of code that performs certain actions and only runs when it is called.

Methods / Functions in C#

- Methods are certain block of code that perform some action.
- Methods are only executed when they are called.
- Methods make maintenance of application easier.
- Methods are also called as functions in C#

```
<Access Specifier> <Return Type> <Method/Function Name>(Parameters)
{
    Method Body (Code for Method Logic)
}
```

Access Specifier - This determines the visibility of a variable or a method from another class.

Return type - The return type is the data type of the value that the method returns.

If the method is not returning any values, then the return type is void.

Method/Function name - This is a unique identifier(name) given to the method, and it is case sensitive.

It cannot be same as any other identifier declared in the class, since the method name must be unique.

Parameters - Parameters are used to pass and receive data from a method.

Parameters always go between round brackets. Parameters are optional and are not always needed.

Method body - All the logic of what the method will do, goes inside the method body.

Methods / Functions in C#

```
<Access Specifier> <Return Type> <Method/Function Name> (Parameters)
{
    Method Body (Code for Method Logic)
}
```

Call Method:

HelloWorld();

Methods / Functions in C#

```
<Access Specifier> <Return Type> <Method/Function Name> (Parameters)
{
    Method Body (Code for Method Logic)
}
```

```
public double Add (double number1, double number2)
{
         double total = number1 + number2;
         return total;
}
Call Method:
Add(10,20);
```

C#

Conditional Flow

Conditional Flow in C#

- IF/ELSE statement
- SWITCH statements
- Ternary Operator

IF/ELSE statement

```
if (condition)
{
     //some code to be executed if condition is valid.
}
```

IF/ELSE statement

IF/ELSE statement

```
if (condition)
{
          //some code to be executed if condition is valid.
}
else
{
          //some code to be executed if condition is NOT valid.
}
```

IF/ELSE statement

```
int marks = 99;
if (marks>90)  //Condition checks if marks is greater than 90
{
         Console.WriteLine("Your Grade - A");
}
else    //If condition is not valid
{
         Console.WriteLine("You missed A grade!");
}
```

Nested IF/ELSE statement

```
if (condition1)
         if (condition2)
```

Nested IF/ELSE statement

```
int marks = 99;
if (marks>90)  //Condition checks if marks is greater than 90
        Console.WriteLine("Your Grade - A");
        if(marks<33) // marks <=90 AND marks <33</pre>
                 Console.WriteLine("You failed.");
                 Console.WriteLine("You missed A grade!");
```

SWITCH statement

SWITCH statement

```
int dayOfWeek = 1; //0 - Sunday , 1 -Monday etc.
switch(dayOfWeek)
            case 0:
                        Console.WriteLine("It's Sunday");
                        break;
            case 1:
                        Console.WriteLine("It's Monday");
                        break;
            case 7:
                        Console.WriteLine("It's Saturday!");
                        break;
            default:
                        Console.WriteLine("Invalid Input!");
                        break;
```

"Its Monday!"

SWITCH statement

```
int dayOfWeek = 100; //0 - Sunday , 1 -Monday etc.
switch(dayOfWeek)
            case 0:
                        Console.WriteLine("It's Sunday");
                        break;
            case 1:
                        Console.WriteLine("It's Monday");
                        break;
            case 7:
                        Console.WriteLine("It's Saturday!");
                        break;
            default:
                        Console.WriteLine("Invalid Input!");
                        break;
```

"Invalid Input!"

C# Loops

Loops in C#

- while loop
- do while loop
- for loop
- foreach loop

While Loop

```
while (condition)
{
     // code to be executed if condition is true
}
```

While Loop

```
int i=0;
while (i<3)
{
    Console.WriteLine(i);
    i++;
}</pre>
Output

0
1
2
```

Do-While Loop

```
do
{
    // code to be executed unless condition is false
    // checks condition after first execution
}
while (condition)
```

Do-While Loop

```
int i=0;
do
{
    Console.WriteLine(i);
    i++;
}
while (i<3)</pre>
Output

0
1
2
```

Do-While Loop

```
int i=100;
do
{
        Console.WriteLine(i);
        i++;
}
while (i<3)</pre>
Output

100
```

For Loop

```
for(statement 1; statement 2; statement 3)
{
    // code to be executed unless condition is false
}
```

- Statement 1 Executed only once, before the execution of code block.
 Contains variable for condition.
- Statement 2 Condition for executing code block
- Statement 3 Executed every time after the code block, contains increment/decrement.

For Loop

```
Executed only once, before the execution of code block. Contains variable for condition.
```

Condition until which loop is executed

Executed every time after the code block, contains increment/decrement

```
for(int i=0;i<3;i++)
{
     Console.WriteLine(i);
}</pre>
```

Output

0 1 2

Foreach Loop

```
foreach(statement)
{
     // code to be executed if the statement is valid
}
```

Statement - This is a unique statement in which we iterate through a list/collection and for each item in that list we perform some operation.

Foreach Loop

```
string name="Ben";
foreach(char c in name)
{
    Console.WriteLine(c);
}
```

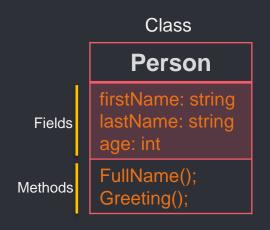


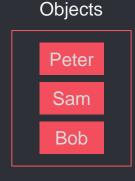
Classes

A class is like a blueprint of a specific object.

Classes in C#

- Class and Object are the basic concepts of Object-Oriented Programming which revolve around the real-life entities.
- A class combines the fields(variables) and methods(functions) int a single unit.
- Core building block of any application.





Classes in C#

```
public class Person
   public string firstName;
   public string lastName;
   public int age;
   public void Greeting()
       Console.WriteLine($"Welcome, {firstName} {lastName}");
   public string FullName()
       return firstName + lastName;
```

Declaring Class

```
Modifier
      public class Person
         public string firstName;
         public string lastName;
         public int age;
         public void Greeting()
             Console.WriteLine($"Welcome, {firstName} {lastName}");
          public string FullName()
             return firstName + lastName;
```

Blue Print / Skeleton



Object

 $\label{problem} \textbf{Basic unit of Object-Oriented Programming and represents the real-life entities.}$

Declaring Objects (Instantiating a class)

- In c#, **Object** is an instance of a **class** and that can be used to access the fields and functions of a **class**.
- Object is a runtime entity, it is created at runtime.

```
Object 1
                                                                               Class
Person sam = new Person();
                                                                               public class Person
sam.firstName= "Sam";
                                                                                  public string firstName;
sam.lastName= "Patrick";
                                                                                  public string lastName;
sam.age= "25";
                                                                                  public int age;
sam.Greeting(); //Prints "Welcome, Sam Patrick"
                                                                                  public void Greeting()
                                                                                     Console.WriteLine($"Welcome, {firstName} {lastName}");
Person ben = new Person();
                                                                                  public string FullName()
sam.firstName= "Ben";
                                                                                     return firstName + lastName;
sam.lastName= "Spark";
sam.age= "20";
sam.Greeting(); //Prints "Welcome, Ben Spark"
```

Object 2

Declaring Objects (Instantiating a class)

- In c#, **Object** is an instance of a **class** and that can be used to access the fields and functions of a **class**.
- Object is a runtime entity, it is created at runtime.

```
Object 1
                                                                               Class
Person sam = new Person();
                                                                               public class Person
sam.firstName= "Sam";
                                                                                  public string firstName;
sam.lastName= "Patrick";
                                                                                  public string lastName;
sam.age= "25";
                                                                                  public int age;
sam.Greeting(); //Prints "Welcome, Sam Patrick"
                                                                                  public void Greeting()
                                                                                     Console.WriteLine($"Welcome, {firstName} {lastName}");
Person ben = new Person();
                                                                                  public string FullName()
sam.firstName= "Ben";
                                                                                     return firstName + lastName;
sam.lastName= "Spark";
sam.age= "20";
sam.Greeting(); //Prints "Welcome, Ben Spark"
```

Object 2



Constructors

A constructor is a **special method** that is used to initialize objects.

Parameterless Constructor (Default)

- A constructor is a method whose name is the same as the name of its type
- If you don't provide a constructor for your class, C# creates one by default

```
public string firstName;
public string lastName;
public int age;
    Console. WriteLine ("Constructor Called");
public void Greeting()
    Console.WriteLine($"Welcome, {firstName} {lastName}");
public string FullName()
    return firstName + lastName;
```

```
Person ben = new Person();
// outputs – "Constructor Called"
```

```
Person ben;
// No Output! Constructor is not called
```

```
Person ben;
Ben = new Person();
// outputs - "Constructor Called"
```

Parameterized Constructor

- A constructor is a method whose name is the same as the name of its type
- If you don't provide a constructor for your class, C# creates one by default

```
public string firstName;
public string lastName;
public int age;
public Person(string firstNm, string lastNm)
    this.firstName = firstNm;
    this.lastName=lastNm;
    Console.WriteLine("Parameter Constructor Called");
public void Greeting()
    Console.WriteLine($"Welcome, {firstName} {lastName}");
public string FullName()
    return firstName + lastName;
```

```
Person ben = new Person("Ben","Spark");
// outputs – "Parameter Constructor Called"
```

```
Person ben;
// No Output! Constructor is not called
```

```
Person ben;
Ben = new Person();
// will not work! Since we do not have default
constructor
```

Multiple Constructor

- A constructor is a method whose name is the same as the name of its type
- If you don't provide a constructor for your class, C# creates one by default

```
public string firstName;
public string lastName;
public int age;
public Person(string firstName)
    this.firstName = firstNm;
    Console. WriteLine ("First Name Constructor Called");
public Person(string firstNm, string lastNm)
    this.firstName = firstNm;
    this.lastName=lastNm;
    Console.WriteLine("Parameter Constructor Called");
public void Greeting()
    Console. WriteLine($"Welcome, {firstName} {lastName}");
public string FullName()
    return firstName + lastName;
```

```
Person ben = new Person("Ben", "Spark");
// outputs – "Parameter Constructor Called"
```

```
Person ben;
// No Output! Constructor is not called
```

```
Person ben;
Ben = new Person("Ben");
// outputs - "First Name Constructor Called"
```



Destructors / Finalizers

Finalizers (destructors) are used to perform any necessary final clean-up when the object is being destroyed.

Destructors / Finalizers

- Finalizers/Destructors are called at the very end when the object is being destroyed.
- Finalizers/Destructors cannot be overloaded.
- Finalizers/Destructors cannot be called. They are invoked automatically.
- Finalizers/Destructors cannot have parameters.

Destructor/Finalizer

```
public class Person
{
    public string firstName;
    public string lastName;
    public int age;
    public void Greeting()
    {
        Console.WriteLine($"Welcome, {firstName} {lastName}");
    }
    public string FullName()
    {
        return firstName + lastName;
    }
    ~Person()
    {
        Console.WriteLine("Destructor Called");
    }
}
```

```
Person ben = new Person("Ben","Spark");
// outputs - "Destructor Called"
```

```
Person ben;
// No Output!
```

(66)

Quotations are commonly printed as a means of inspiration and to invoke philosophical thoughts from the reader.

Single-line Comment

// Single line comment example int x=100;

66

Single-line Comment

```
/*
Multi-line comment
example
*/
int x=100;
```

THIS IS A SLIDE TITLE

- Here you have:
 - A list of items
 - And some text
 - But remember not to overload your slides with content

Your audience will listen to you or read the content, but won't do both.



BIG CONCEPT

Bring the attention of your audience over a key concept using icons or illustrations

YOU CAN ALSO SPLIT YOUR CONTENT

White

Is the color of milk and fresh snow, the color produced by the combination of all the colors of the visible spectrum.

Black

Is the color of ebony and of outer space. It has been the symbolic color of elegance, solemnity and authority.

IN TWO OR THREE COLUMNS

Yellow

Is the color of gold, butter and ripe lemons. In the spectrum of visible light, yellow is found between green and orange.

Blue

Is the colour of the clear sky and the deep sea. It is located between violet and green on the optical spectrum.

Red

Is the color of blood, and because of this it has historically been associated with sacrifice, danger and courage.

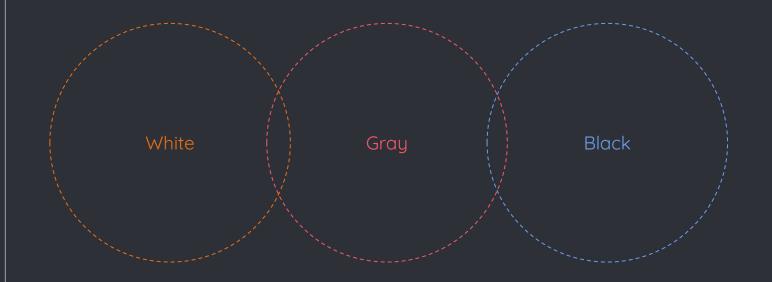
A PICTURE IS WORTH A THOUSAND WORDS



A complex idea can be conveyed with just a single still image, namely making it possible to absorb large amounts of data quickly.



USE CHARTS TO EXPLAIN YOUR IDEAS



AND TABLES TO COMPARE DATA

	A	В	С
Yellow	10	20	7
Blue	30	15	10
Orange	5	24	16

MAPS



89,526,124

Whoa! That's a big number, aren't you proud?

89,526,124\$That's a lot of money

185,244 users

And a lot of users

100%

Total success!

OUR PROCESS IS EASY

First

Second

Last

LET'S REVIEW SOME CONCEPTS



Yellow

Is the color of gold, butter and ripe lemons. In the spectrum of visible light, yellow is found between green and orange.



Yellow

Is the color of gold, butter and ripe lemons. In the spectrum of visible light, yellow is found between green and orange.



Blue

Is the colour of the clear sky and the deep sea. It is located between violet and green on the optical spectrum.



Blue

Is the colour of the clear sky and the deep sea. It is located between violet and green on the optical spectrum.



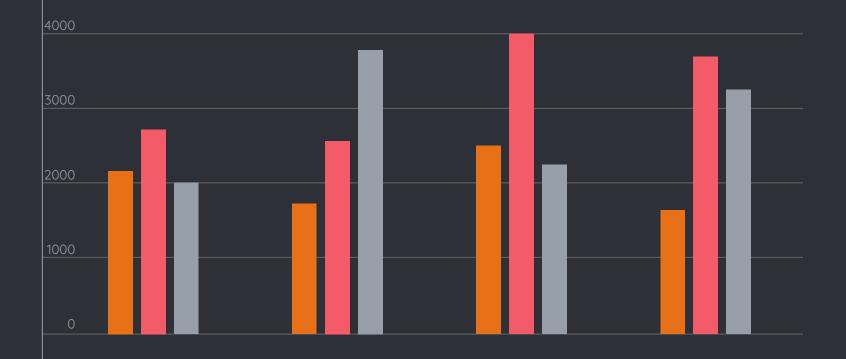
Red

Is the color of blood, and because of this it has historically been associated with sacrifice, danger and courage.



Red

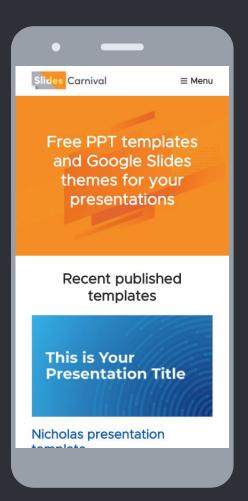
Is the color of blood, and because of this it has historically been associated with sacrifice, danger and courage.



You can copy&paste graphs from Excel or Google Sheets

MOBILE PROJECT

Show and explain your web, app or software projects using these gadget templates.



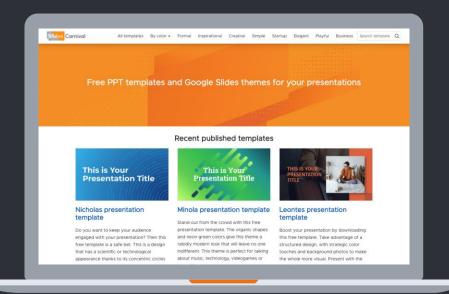
TABLET PROJECT

Show and explain your web, app or software projects using these gadget templates.



DESKTOP PROJECT

Show and explain your web, app or software projects using these gadget templates.



Thanks!

ANY QUESTIONS?

You can find me at @username user@mail.me

WHILE LOOP (ASSIGNMENT)

- Get input from user (integer).
- O Display number from input number to 0.
- o If user enters 5, output should be the following
 - . -
 - □ 4

 - □ 1

WHILE LOOP (ASSIGNMENT)

- O User has to build their coffee
- User will get 3 options
 - Cream
 - Sugar
 - Expresso
 - Done
- Once the coffee is ready, they will select option 4
 (done) and we will display their final selection
- After they make any selection, we will display their current selection

DO-WHILE LOOP (ASSIGNMENT)

- O Get input from user (integer).
- O Display number from input number to 0.
- o If user enters 5, output should be the following
 - _ 5
 - ⁻ 4
 - **-** 3

DO-WHILE LOOP (ASSIGNMENT)

- User has to build their coffee
- User will get 3 options
 - Cream
 - Sugar
 - Expresso
 - Done
- Once the coffee is ready, they will select option 4 (done) and we will display their final selection
- After they make any selection, we will display their current selection

FOR LOOP (ASSIGNMENT)

- O Get input from user (integer).
- O Display number from input number to 0.
- o If user enters 5, output should be the following
 - _ 3
 - **-** 4

 - **□** 1

• FOR LOOP

****	5	0	****	5	5	****	***	**	5
****	4	1	****	4	4	****	2 **	**	4
***	3	2	***	3	3	***	4 *	**	3
**	2	3	**	2	2	**	6	**	2
*	1	4	*	1	1	*	8	*	1

FOR LOOP - ASSIGNMENT

**** ****
*** ***
** **
* *
** **
*** ***
**** ****

PRESENTATION DESIGN

- This presentations uses the following typographies:
 - Titles & body copy: Quicksand

Download for free at:

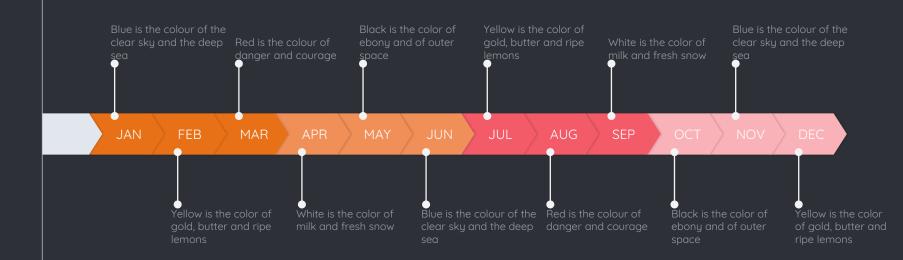
https://www.fontsquirrel.com/fonts/quicksand

You don't need to keep this slide in your presentation. It's only here to serve you as a design guide if you need to create new slides or download the fonts to edit the presentation in PowerPoint®

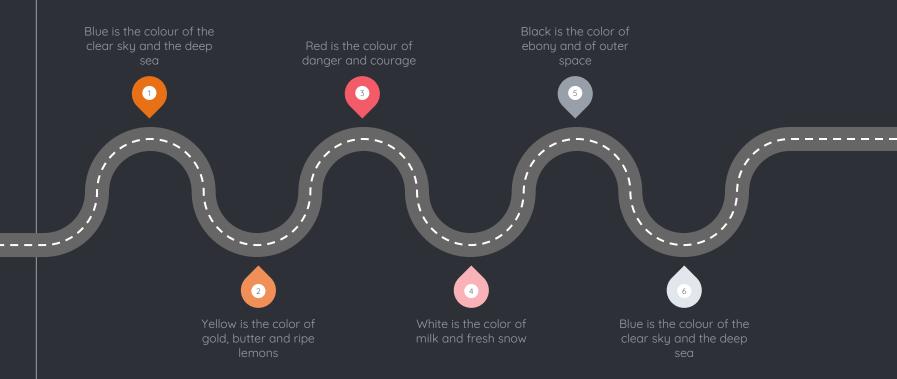


For Business Plans, Marketing Plans, Project Proposals, Lessons, etc

Timeline



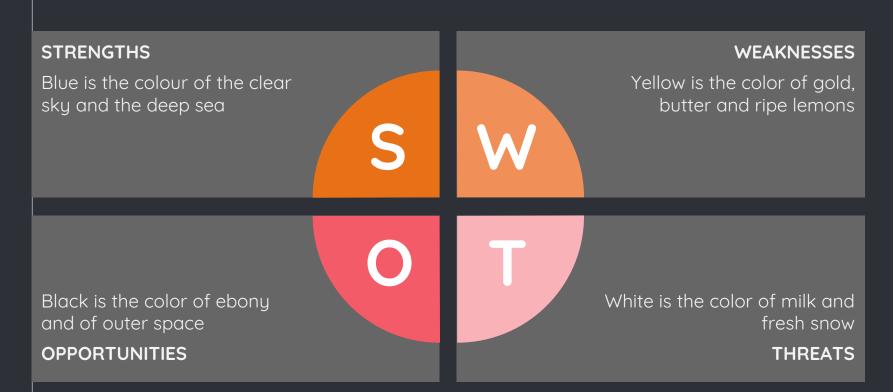
Roadmap



Gantt chart

	Week 1							Week 2						
	1	2	3	4	5	6	7		9	10	11	12	13	14
Task 1														
Task 2														
Task 3														
Task 4											•			
Task 5														
Task 6														
Task 7														
Task 8														

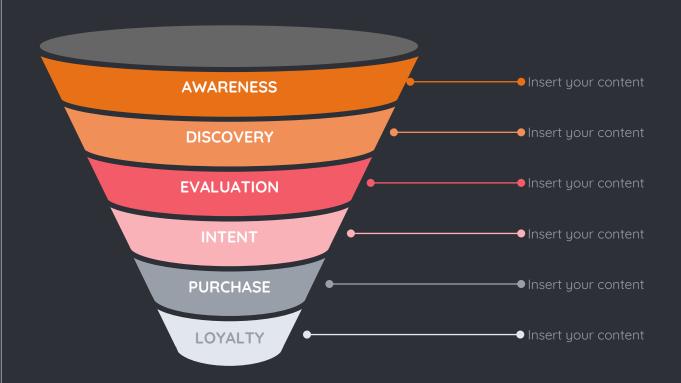
SWOT Analysis



Business Model Canvas

Key Partners	Key Activities	Value Propo	sitions	Customer Relationships	Customer Segments	\mathcal{L}
Insert your content	Insert your content	Insert your cor	tent	Insert your content	Insert your content	
	Key Resources Insert your content			Channels Insert your content		
Cost Structure			Revenue Str	eams		<u>~~</u>
Insert your content			Insert your cor	ntent		

Funnel



Team Presentation



Imani Jackson

JOB TITLE

Blue is the colour of th

Blue is the colour of the clear sky and the deep sea



JOB TITLE

Blue is the colour of the clear sky and the deep sea

Marcos Galán



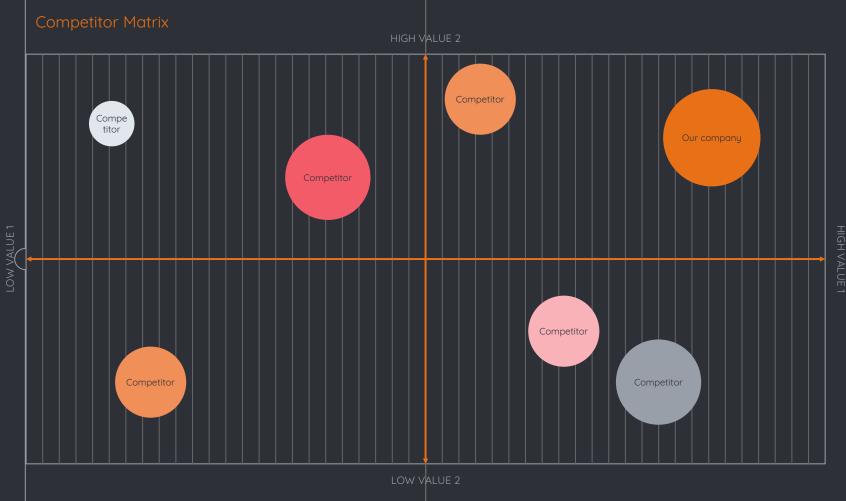
JOB TITLE

Blue is the colour of the clear sky and the deep sea

Ixchel Valdía



Nils ÅrudJOB TITLE
Blue is the colour of the
clear sky and the deep sea



Weekly Planner

	SUNDAY	MONDAY	TUESDAY	WEDNESDAY	THURSDAY	FRIDAY	SATURDAY
09:00 - 09:45	Task						
10:00 - 10:45	Task						
11:00 - 11:45	Task						
12:00 - 13:15	√ Free time						
13:30 - 14:15	Task						
14:30 - 15:15	Task						
15:30 - 16:15	Task						



SlidesCarnival icons are editable shapes.

This means that you can:

- Resize them without losing quality.
- Change line color, width and style.

sn't that nice?:)

Examples:







Diagrams and infographics



Now you can use any emoji as an icon!

And of course it resizes without losing quality and you can change the color.

How? Follow Google instructions https://twitter.com/googledocs/status/730087240156643328





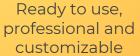
Free templates for all your presentation needs



For PowerPoint and Google Slides



100% free for personal or commercial use



Blow your audience away with attractive visuals