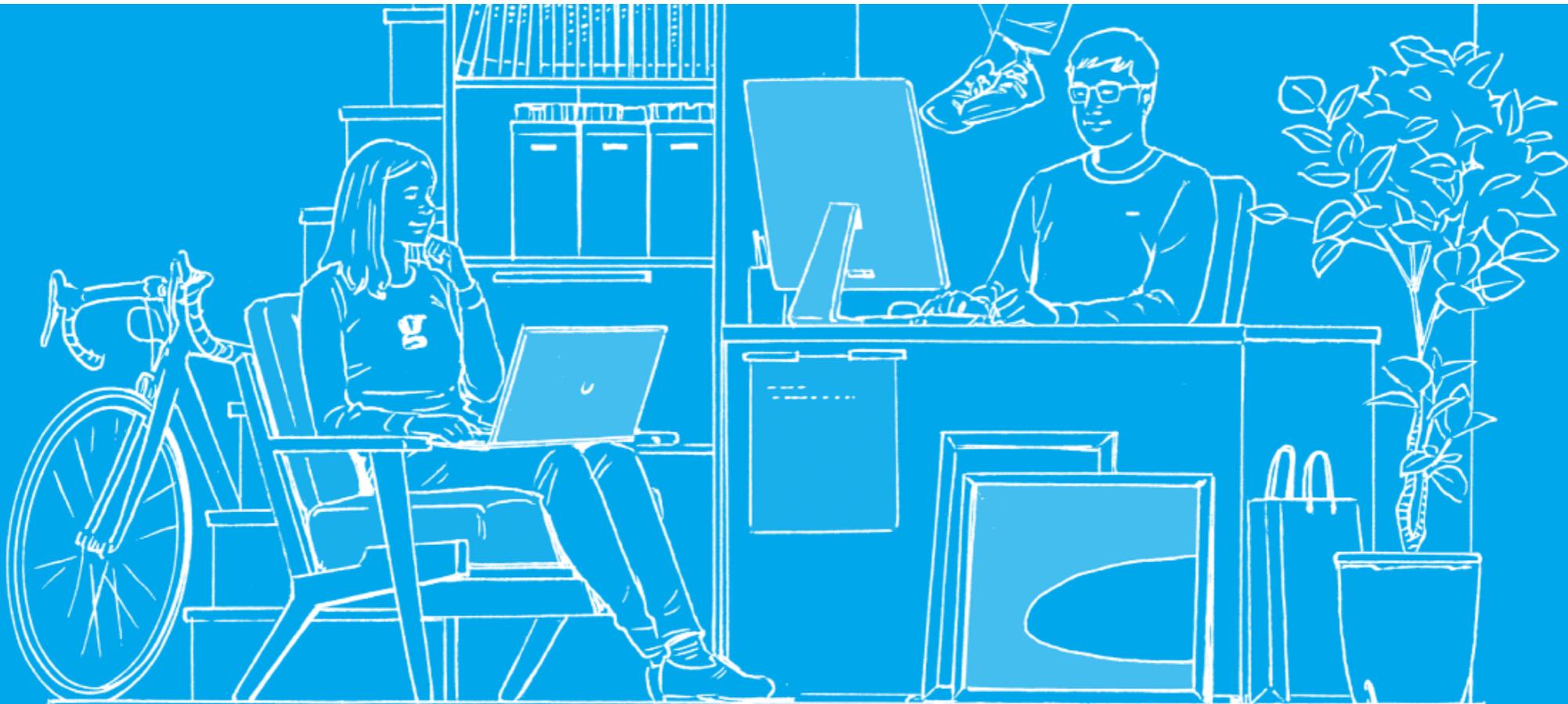




**G's ACADEMY  
TOKYO**

# Gs React 認証



# React+認証

完成コード以下

[https://github.com/mitunori/  
react\\_auth/](https://github.com/mitunori/react_auth/)

# 認証事前設定

# 設定

プロジェクトの概要 |

Project shortcuts

- Firestore Database
- Authentication
- Functions
- Hosting

プロダクトのカテゴリ

構築

- Authentication
- App Check
- Firestore Database
- Realtime Database
- Extensions
- Storage
- Hosting
- Functions

## Authentication

サーバーサイドのコードを使わずに、さまざまなプロバイダのユーザーを認証し管理します

[始める](#)

詳細

開始方法  
ドキュメントを表示

Introducing Firebase

# 設定

## Authentication

Users    **Sign-in method**    Templates    Usage    Settings

ログイン プロバイダ

ログイン方法を追加して Firebase Auth の利用を開始しましょう

ネイティブのプロバイダ

✉ メール / パスワード

📞 電話番号

👤 匿名

追加のプロバイダ

Google

Game Center

Microsoft

Facebook

Apple

Twitter

Play ゲーム

Github

Yahoo!

カスタム プロバイダ

🔒 OpenID Connect

🔒 SAML

# 設定



有効にする

Google ログインが、接続された Apple アプリおよびウェブアプリ上で自動的に設定されます。ご自身の Android アプリに Google ログインを設定するには、[プロジェクトの設定](#)でアプリごとに [SHA1 フィンガープリント](#) を追加する必要があります。

以下の[プロジェクト レベル設定](#)を更新して次に進む

プロジェクトの公開名 ?

project-107164342202

プロジェクトのサポートメール ?

mitukun32@gmail.com

外部プロジェクトのクライアント ID をセーフリストに追加します  
(オプション)

ウェブ SDK 構成 ?

キャンセル

保存

# 設定

## ログイン プロバイダ

プロバイダ



ステータス



有効

新しいプロバイダを追加

# 設定

```
src > JS firebase.js > ...
1  //Firebase ver9 compliant (modular)
2  import { initializeApp } from "firebase/app";
3  import { getAuth } from "firebase/auth";
4  import { getFirestore } from "firebase/firestore";
5
6  const firebaseApp = initializeApp({
7    apiKey: import.meta.env.VITE_apiKey,
8    authDomain: import.meta.env.VITE_authDomain,
9    projectId: import.meta.env.VITE_projectId,
10   storageBucket: import.meta.env.VITE_storageBucket,
11   messagingSenderId: import.meta.env.VITE.messagingSenderId,
12   appId: import.meta.env.VITE_appId,
13 });
14
15 //Firebase ver9 compliant (modular)
16 export const auth = getAuth(firebaseApp);
17 export const db = getFirestore(firebaseApp);
18 |
```

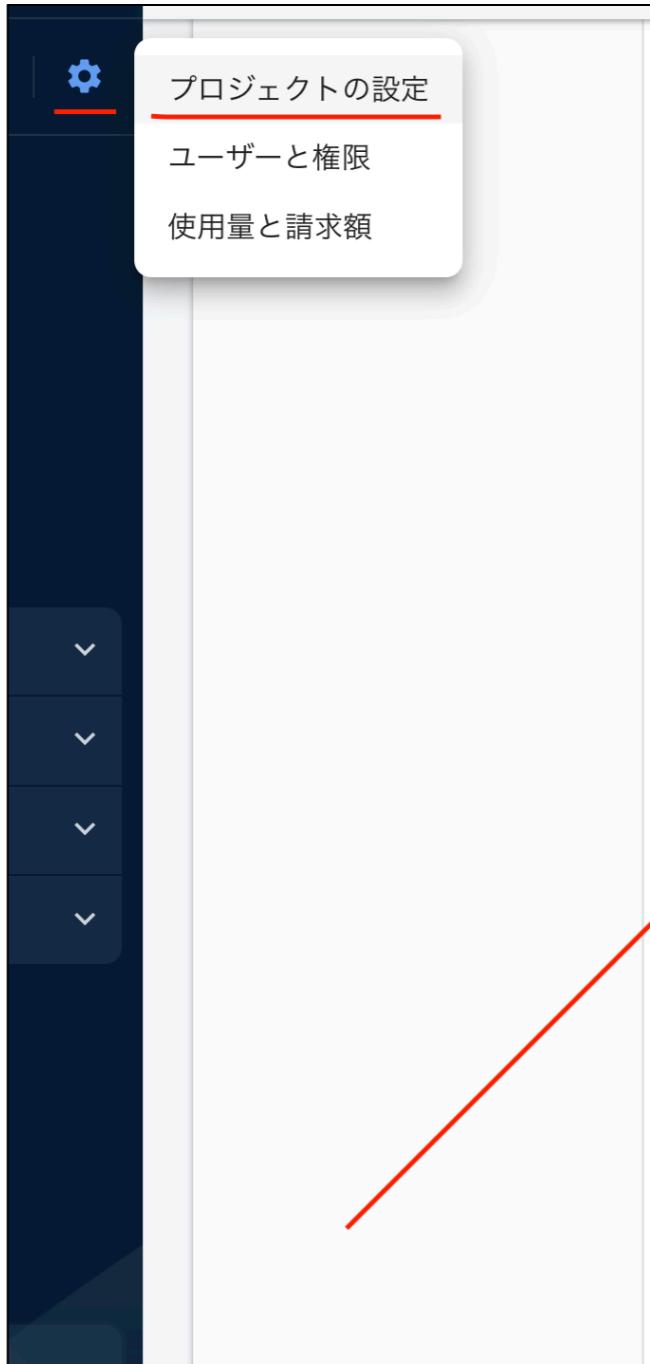
src/firebase.jsを作成し、中身をこのようにする（今まで作成したものでOK）

# 設定

```
📁 .env
1  VITE_apiKey="AIzaSyBo4oHBsexIrBgM880jhbLCVvoPSqq_qtA"
2  VITE authDomain="react-future-d8bb5.firebaseio.com"
3  VITE projectId="react-future-d8bb5"
4  VITE storageBucket="react-future-d8bb5.appspot.com"
5  VITE messagingSenderId="107164342202"
6  VITE appId="1:107164342202:web:b1b3964ffs"
```

.envを作成し、中身はfirebaseのプロジェクトの設定に記述されているものに変更  
→次のページに続く

# 設定



npm とモジュールバンドラ (webpack や Rollup など) をすでに使用している場合は、次のコマンドを実行して最新の SDK をインストールできます。

```
$ npm install firebase
```

次に Firebase を初期化し、使用するプロダクトの SDK の利用を開始します。

```
// Import the functions you need from the SDKs you need
import { initializeApp } from "firebase/app";
// TODO: Add SDKs for Firebase products that you want to use
// https://firebase.google.com/docs/web/setup#available-libraries

// Your web app's Firebase configuration
const firebaseConfig = {
  apiKey: "AIzaSyBo4oHBsexIrBgM880jhbLCVvoPSqq_qtA",
  authDomain: "react-future-d8bb5.firebaseio.com",
  projectId: "react-future-d8bb5",
  storageBucket: "react-future-d8bb5.appspot.com",
  messagingSenderId: "107164342202",
  appId: "1:107164342202:web:b1b3964ff5be128f422f83"
};

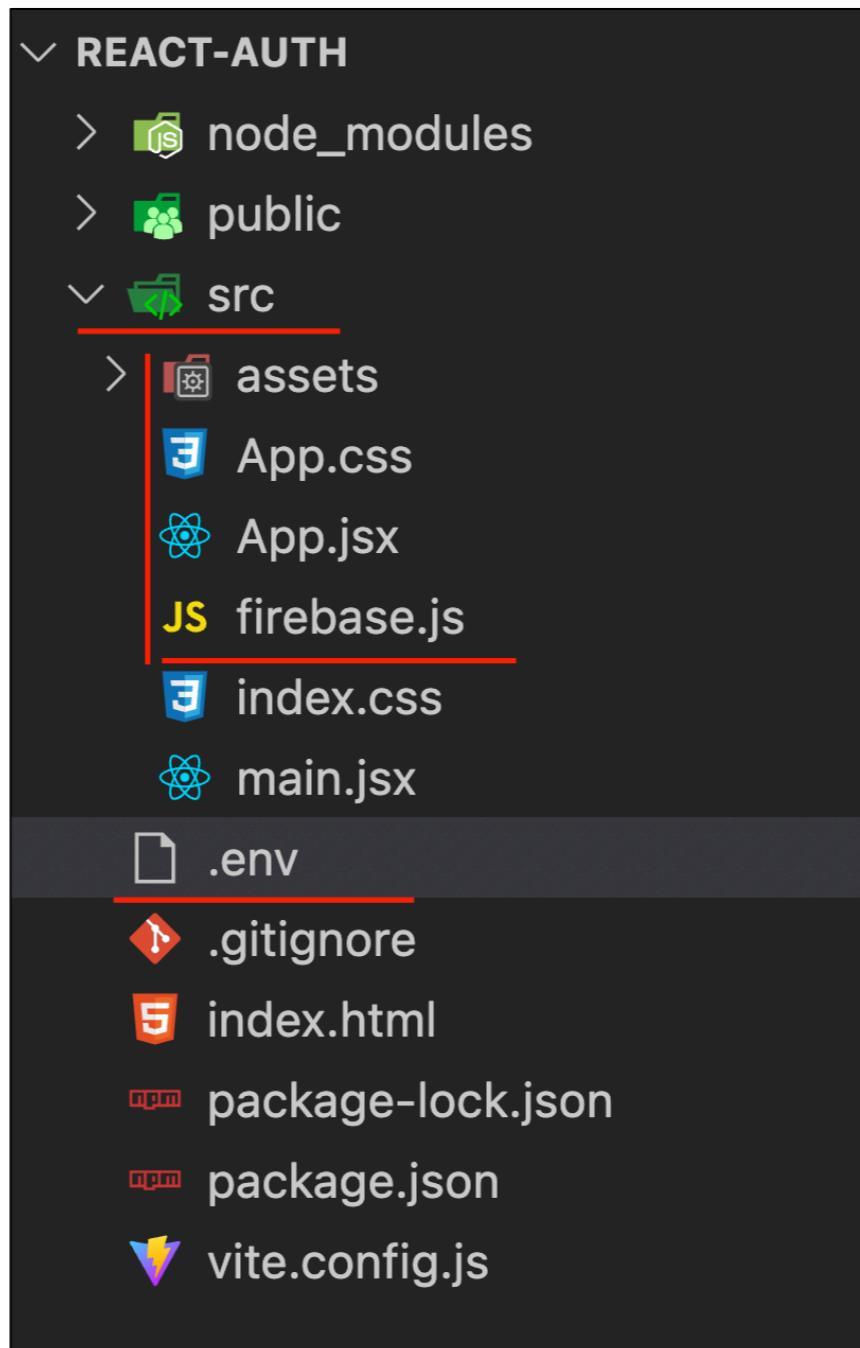
// Initialize Firebase
const app = initializeApp(firebaseConfig);
```

注: このオプションではモジュラー JavaScript SDK を使用します。これにより SDK のサイズが小さくなります。

ウェブ向け Firebase の詳細については、こちらをご覧ください: [使ってみる](#)、[ウェブ SDK API リファレンス](#)、[サンプル](#)

.envの中身はこちらになります😊

# 設定



.envとfirebase.jsの場所が違うので注意しましょう😊

# 設定

## Authentication

Sign-in method   Templates   Usage   **Settings**

設定

ユーザー アカウント管理

 ユーザー アカウントのリンク

 ユーザー アクション

 ブロッキング関数

 ユーザー アクティビティのログイン

 申し込みによる割り当て

ドメイン

 承認済みドメイン

承認済みドメイン

セキュリティ上の理由により、電話による認証、Google 認証、またはサードパーティの認証を使用するには、ドメインが OAuth リダイレクトに対して承認されている必要があります。[詳細](#)

ドメインの追加

承認済みドメイン

種類

localhost

Default

react-future-d8bb5.firebaseio.com

Default

react-future-d8bb5.web.app

Default

127.0.0.1

Custom



viteはこの設定をしないとログインが成功しないので要注意！

# ターミナルで以下を行う

1.

**npm i firebase**

2.

**npm i react-router-dom**

必要な機能をインストールし、**package.json**に追加されていればOKです😊

# コード実装

# コード記述

The screenshot shows a file explorer on the left and a code editor on the right. The file explorer displays a project structure for a React application named 'REACT-AUTH'. The 'src' directory contains several files and folders: 'node\_modules', 'public', 'assets', 'components' (which is currently selected), 'Login.jsx' (which is also highlighted), 'Success.jsx', 'App.css', 'App.jsx', 'firebase.js', 'index.css', 'main.jsx', '.env', '.gitignore', and 'index.html'. The code editor shows the path 'src > components > Login.jsx' and the first character '1 |' of the file content. To the right of the code editor, there is explanatory text in red: 'components/配下に Login.jsx Success.jsx の2つのファイルを作成する' (Create two files, Login.jsx and Success.jsx, in the components/ directory).

```
src > components > Login.jsx
1 |
```

components/配下に  
Login.jsx  
Success.jsx  
の2つのファイルを作成する

react のショートカットを活用してファイルの雛形を作成しましょう😊

# コード記述

```
src > components > Login.jsx > ...
1 import React from "react";
2
3 const Login = () => {
4   return <div>Login</div>;
5 }
6
7 export default Login;
8 |
```

```
src > components > Success.jsx > ...
1 import React from "react";
2
3 const Success = () => {
4   return <div>Success</div>;
5 }
6
7 export default Success;
8 |
```

このように雛形のファイルを作成しましょう😊

# コード記述

```
Atom App.jsx X  
src > App.jsx > ...  
1 import "./App.css";  
2 import Success from "./components/Success";  
3 import Login from "./components/Login";  
4 import { BrowserRouter, Routes, Route } from "react-router-dom";  
5  
6 function App() {  
7   return (  
8     <div className="App">  
9       <BrowserRouter>  
10      <Routes>  
11        <Route path="/" element={<Success />} />  
12        <Route path="/login" element={<Login />} />  
13      </Routes>  
14    </BrowserRouter>  
15  </div>  
16);  
17}  
18  
19 export default App;  
20
```

App.jsxを変更します😊

# Loginコンポーネント実装

# Loginコンポーネント

```
src > components > Login.jsx > Login
1 import React, { useEffect } from "react";
2 import { useNavigate } from "react-router-dom";
3 import { signInWithPopup, GoogleAuthProvider } from "firebase/auth";
4 import { auth } from "../firebase";
5 import { onAuthStateChanged } from "firebase/auth";
6
```

必要なものをimportをしておきましょう😊

# Loginコンポーネント

```
const Login = () => {
  // ページ遷移をさせる記述
  const navigate = useNavigate();

  // グーグル認証のコード
  const googleLogIn = () => {
    const provider = new GoogleAuthProvider();

    console.log(auth, "認証情報");
    signInWithPopup(auth, provider)
      .then((result) => {
        console.log(result, "成功したらresultにデータが入ります😊");
        console.log("Googleアカウントでログインしました。");
      })
      .catch((error) => {
        console.log(error);
      });
  };
};
```

ページ遷移をさせる記述、グーグル認証コードを記述しましょう😊

# Loginコンポーネント

```
useEffect(() => {
  const unSub = onAuthStateChanged(auth, (user) => {
    console.log(user, "user情報をチェック！");
    //userにはログインor登録されているかの状態がtrue/falseで入ってくるので、!userはfalse=user情報がないとき！
    if (user) {
      user && navigate("/");
    } else {
      user && navigate("/login");
    }
  });
  return () => unSub();
}, [navigate]);

return (
  <div>
    <h2>GoogleLogin</h2>
    <button onClick={googleLogIn}>ログイン</button>
  </div>
);
};

export default Login;
```

useEffectの箇所、jsx内のhtmlを記述しましょう😊

# ポイント

```
useEffect(() => {
  const unSub = onAuthStateChanged(auth, (user) => {
    console.log(user, "user情報をチェック！");
    //userにはログインor登録されているかの状態がtrue/falseで入ってくるので、 !userはfalse=user情報がないとき！
    if (user) {
      user && navigate("/");
    } else {
      user && navigate("/login");
    }
  });
  return () => unSub();
}, [navigate]);
```

**onAuthStateChanged**というおまじないを使うことで常に、「ログイン」状態を取得することができます😊

**useEffect**を使って、画面が表示された直後に「ログイン」状態を判断し  
情報がある→成功なので、**navigate('/')**でページを遷移させています😊  
情報がない→ログインしていないので、**navigate('/login')**ページに遷移させています😊

# Successコンポーネント の実装

# Successコンポーネント

```
src > components > 🚀 Success.jsx > ...
1 import React, { useEffect } from "react";
2 import { useNavigate } from "react-router-dom";
3 import { onAuthStateChanged, signOut } from "firebase/auth";
4 import { auth } from "../firebase";
5
```

必要なものをimportをしておきましょう😊

# Successコンポーネント

```
const Success = () => {
  const navigate = useNavigate();

  // 追加
  useEffect(() => {
    const unSub = onAuthStateChanged(auth, (user) => {
      console.log(user, "user情報をチェック！");
      //userにはログインor登録されているかの状態がtrue/falseで入ってくるので、 !userはfalse=user情報がないとき！
      !user && navigate("/login");
    });
    return () => unSub();
  }, [navigate]);

  const googleLogOut = async () => {
    try {
      await signOut(auth);
      navigate("/login");
    } catch (error) {
      alert(error.message);
    }
  };
}
```

onAuthStateChangedを使って「ログイン」しているかを判断してページの表示を決定しています😊

# Successコンポーネント

```
return (
  <div>
    <h1>Success</h1>
    <div>成功した時に表示したい中身を記述してみましょう😊</div>
    {/* ログアウトのボタン */}
    <button onClick={googleLogOut}>ログアウト</button>
  </div>
);
};

export default Success;
```

成功しているときは、Successコンポーネントが表示されるので、自分の好きなものを表示してみましょう😊

# Successコンポーネント

```
const googleLogOut = async () => {
  try {
    await signOut(auth);
    navigate("/login");
  } catch (error) {
    alert(error.message);
  }
};
```

ログアウトのボタンを押す→googleLogOutが実行されます😊

その際にfirebase側が用意してある[signOut]というおまじないを実行します



# 設定

## Authentication

Users Sign-in method Templates Usage Settings

メールアドレス、電話番号、またはユーザー UID で検索					ユーザーを追加	C	:
ID	プロバイダ	作成日 ↓	ログイン日	ユーザー UID			
mitukun32@gmail.com	G	2022/07/31	2022/07/31	uHXTAn7C2GeCjNVQYDPOQcoOX...			

ログインの処理が成功するとこのように、管理画面上で反映されています😊

# まとめ

認証を理解する(v9は特に注意！)

onAuthStateChangedを活用することでページの表示を切り替えられます

1.必要な情報をimportすることを忘れない

2.説明書（ドキュメントやリファレンス）など

は必ず見ましょう！

3. .envがないと動かないので注意しましょう！

認証を体感していただきました 😊

この機能を活用することでログイン機能を簡単に実装できますので有効活用してみましょう

う 😊

\*なるべくわかりやすい表現やかなり噛み碎いた言い回しにしていますが、あくまでも一つの意見として参考にしてみてください  
違う方や他の記事で色々な解説や説明などもありますので、それらと見比べながらやってみると理解が深まるかと思います