**React**

JavaScript library for user interface designing

Declarative

Component based-class based and Functional component

Consist of JSX

Pre-requisite

Intall node js

To install react

npx create-react-app my-app

To run application

npm start

simple code:

import ‘./App.css’;

const App=()=> {

return(

<div className=”App”>

<h1>Hello {name}!</h1>

</div>

);

}

}

export default App;

React fragment used to render multiple/different element of html

{

name?(

<> 🡨 react fragment

test

</>

): (

<>

<h1>test</h1>

<h2>test2</h2>

</>

)

}

Creating a functional component

Const Person = () => { //component name to be started with capital

Return (

<>

<h1>test</h1>

<h2>test2</h2>

<h1>test3</h1>

<h2>test4</h2>

</>

)

}

const App=()=> {

return(

<div className=”App”>

<Person />

<Person />

<Person />

</div>

);

}

}

Props

Allow u to pass navig data through react components. Passed via attributes.

Const Person = (props)=> {

Return (

<>

<h1>Name: {props.name}</h1>

<h2>Last Name: {props.lastName}</</h2>

<h2>Age: {props.age}</</h2>

</>

)

}

const App=()=> {

return(

<div className=”App”>

<Person name={‘John’} lastName={‘Doe’} age={25}/>

<Person />

<Person />

</div>

);

}

}

State

Plain JavaScript object represent a piece of information about the components current situation

Pre-requiste

import {useState} from ‘react’;

Hook

Whenever we start a function and we start with use then it is called hook

import {useState} from ‘react’;

import ‘./App.css’;

const App=()=> {

const [counter, setCounter] = useState(0);

useEffect(()=>{

setCounter(100);

},[]);

return(

<div className=”App”>

<button onclick={()=> setCounter((prevCount)=> prevCount -1)}>-</button>

<h1>{counter}</h1>

<button onclick={()=> setCounter((prevCount)=> prevCount +1)}>+</button>

</div>

);

}

}

export default App;

Building application

App.css-from git

search.svg-form git

Index.js

import React from ‘react’;

import ReactDOM form ‘react-dom’;

import App from ‘./App’;

ReactDOM.render(<App/>, document.getElemenyById(‘root’));

MovieCard.jsx-react component

Import React form ‘react’;

Const MovieCard=({movie})=>{//used movie1 instead of props, if props used should be mentionsed everywhere

return(

<div className=”movie”>

<div>

<p>{movie.Year}</p>

</div>

<div>

<img src={movie.Poster}!==’N/A’? movie.Poster: ‘https://via.placeholder.com/400’ alt={movie.Title}/>

</div>

<div>

<span>{movie.Type}</span>

<h3>{movie.Title}</h3>

</div>

</div>

)

}

Export deault MovieCard;

App.js

Import React from ‘react’;

Import ‘./App.css’;

Import SearchIcon from ‘./search.svg’

Import MovieCard from ‘./MovieCard’;

const API\_URL=”https:www.omdbi.com?apikey=num.”;//num to be put when created an account

const movie1={

json from git

}

const App=()=> {

const [movies, setMovies] = useState([]);

const [searchTerm, setSearchTerm] = useSate([]);

const searchMovies = async(title)=> {

const response = await fetch(‘$(API\_URL)’&s=$(title)’);

const data = await response.json();

console.log(data.Search);

}

useEffect(()=> {

searchMovies(‘Spiderman’);

},[]);

return (

<div className=”app”>

<h1>MovieLand</h1>

<div className=”search”>

<input

Placeholder=”Search for movies”

Value={searchTerm}

Onchange={(e)=>setSearchTerm(e.target.value)}

/>

<img

Src={SearchIcon}

Alt=”search”

Onclick={()=>searchMovies(searchTerm)}/>

</div>

{

movies?.length > 0

?(

<div className=”container”>

{movies.map((movie)=>(

<MovieCard movie1={movie}/>

))}

</div>

): (

<div className=”empty”>

<h2>No movies found</h2>

</div>

)

}

/\*

<div className=”container”>

/\*

<div className=”movie”>

<div>

<p>{movie1.Year}</p>

</div>

<div>

<img src={movie1.Poster}!==’N/A’? movie1.Poster: ‘https://via.placeholder.com/400’ alt={movie1.Title}/>

</div>

<div>

<span>{movie1.Type}</span>

<h3>{movie1.Title}</h3>

</div>

</div>

\*/

<MovieCard movie1={movie1}/>

</div>

\*/

</div>

)

}

**Web dev simplified**

Props is passed into a component and props handled outside

State is inside a component and handled inside

State is re-rendered when it changes

Hook-only for function component

Cannot be put inside the conditional statement

useState

it consists of two parameter inside a array as it returns array of two parameter

const [count, setCount]= useState(4)

count is the current state and setCount is the updated state(function) to be set.

Fucntion decre(){

setCount(prevCount=>prevCount-1)

}

onclick={decre} inside a button

instead of 4 we can give an function to only run once, otherwise everytime u render

it renders the value which is not needed

const [count, setCount]= useState(()=> {

console.log(“run function”);

return 4;

})

For merging property when a single property is updated …prevState is to be used

Or can use multiple state

Const [state, setState]= useState({count: 4, theme:’blue’})

Const count = state.count

Const theme = state.theme

Fucntion decrementCount(){

setState(orevState => {  
return {…prevState, count: prevState.count-1}

})

}

useEffect

**Freecodecamp**

Why react

Composable- component based

Declarative- like variables so no need to repeatedly declare

Hireable skill

Actively maintained by skilled people

Props used as function for creating dynamic components. Like youtube home page tiles.

It cannot be just imported as it becomes static without change in data.

Static websites done till now

Dynamic to be done

Event listeners

State

Conditional rendering

Forms

Side effects

function handleClick(){

console.log(“I was clicked!”);

<button onclick={handleClick}>Click me</button>

If handleClick()-> it loads on refreshing page before clicking

Camle case to be followed

<https://legacy.reactjs.org/docs/events.html#mouse-events>

State used to update the data value correctly, react will automatically update the UI part.

Interaction like bookmarking or saving reels in apps. Uses state to update the user response.

**Props vs State**

Props refer to the properties being passed into a component for it to work correctly, like how a function receives parameters: “from above.” A component receiving props is not allowed to modify those props(immutable).

State refers to the values that are managed by the components, like variables declared inside a function. Anytime you have changing values that should be saved/displayed, you’ll likely be using state.

**Code for Employee profile//toggle favourite**

App.js

Function App()

{

Const[contact,setContact]=React.useState({

firstName:”John”,

lastName:”Doe”,

phone:”+1 (719) 555-1212”

email: [itsmyrealname@example.com](mailto:itsmyrealname@example.com)

isFavourite=false

})

Let starIcon= constact.isFavourite? “star-filled.png”:”star-empty.png”

Function toggleFavourite(){

setContact(prev=>{

return{

…prev,

isFavourite=!prev.isFavourite

}

})

}

}

Challenge:

Import Star from ‘./components/Star.js’

<Star

Key={contact.isFavourite}

isFilled={ contact.isFavourite }

/>

Star.js

Import React from ‘react’

Function Star(props)

{

Let starticon= props.isFilled?”star-filled.png”:”star-empty.png”;

Return(

<img src={‘../images/${starticon}’}

className=”class--icon”

)

}

Export default star;

//custom components can’t have dom event listeners

<Star

Key={contact.isFavourite}

isFilled={ contact.isFavourite }

handleclick={toggleFavourite}//passing function as custom properties

/>

Star.js

Import React from ‘react’

Function Star(props)

{

Let starticon= props.isFilled?”star-filled.png”:”star-empty.png”;

Return(

<img src={‘../images/${starticon}’}

className=”class--icon”

onclick={props.handleclick}

)

}

Export default star;

Passing data to components

Keep the state as close to component(local)

To pass to sibling component u need to create state in the parent component and pass down to the children

Forms

Controlled component

To have single truthy value in a components

We are giving react state to control the html values

Side Effects

What are React’s primary tasks?

Work with the DOM/browser to render UI to the page

Manage state for us between render cycles (i.e state values are “remembered” from one render to the next)

Keep the UI updated whenever state changes occur

What can’t React handle on its own?

(Out)side effects

-local Storage

-API/database interactions

-subscription (e.g web sockets)

-syncing 2 different internal states

-basically anything that React isn’t in charge of

useEffect() – a blank canvas provided by react, to interact with the outside effect and to sync with the state

Code that runs because a component was displayed should be in Effects, the rest should be in events

You can fetch data with Effects, but you need to implement cleanup to avoid race conditions.

Putting the function as a callback function inside use Effect will only run after the UI is rendered

UseEffect takes a function as its parameter. If that function return something, it needs to be a cleanup function. Otherwise, it should return nothing. If we make it an async function, it automatically returns a promise instead of a function or nothing. Therefore, if you want to use async operations inside of useEffect, you need to define the function separately inside of useEffect, you need to define the function separately inside of the callback function, as seen below

Section 3-

Event listeners

State

Conditional rendering

Forms

Side effects

**Projects**

1. Dark mode project
2. Note-app project

-sync notes with localStorage

localStorage.getItem(“key”)

localStorage.getItem(“key”,value)

not value must be a string, so if you have a more complex value like an array or object to save, you’ll need to use:

JSON.stringify(value)

To retrieve back: JSON.parse(stringifiedvalue)

-add note summary titles

-move modified noted to the top of the list

-delete notes

**Tenzies project**

**Solo project-quizical**

**Advanced react course**

**React router**

**Css in JS**

**Next.js**

**Performance/optimization**

**More hooks**

**Web Dev – Reacts errors**

1.Instead of useState we can use {useRef}

Import {useRef} from “react”

Function App(){

Const emailRef = useRef()//here instead of state we can use reference as on the final click of the button only we need the value

Const passwordRef = useRef()

Function onSubmit(e){

e.preventDefault()

console.log({

email: emailRef.current.value,

password: passwordRef.current.value,

})

}

Return(

<form onSubmit ={onSubmit}>

<label htmlFor=’email’>Email</label>

<input ref={emailRef} type=”email” id=”email”/>

<label htmlFor=”password”>Password</label>

<input ref={passwordRef} type=”password” id=”password” />

<button type=”submit”>Submit</button>

</form>

)

}

Export default App

2. use of function version for setting the state

Setcount(currentCount => {

Return currenCount + amount

})

3. State does not update immediately

4. use of useEffect when some code dependent on change of other variable is there. It need to be be put inside the useEffect

But not using effects where not necessary is the best practice.

Like for fetching and forms

5.Referential Equality mistakes, as the all states are re-rendered when one state changes. Two objects having same value but initialized differently aren’t the same.

So use of useMemo()

Const person = useMemo(()=>{

Return {age, name}

},[age,name])

6.Not aborting fetch requests

Import {useEffect, useState} from ‘react’

Export function useFetch(url){

Const [loading, setLoading] = useState(true)

Const [data,setData]= useState()

Const [error, setError]=useState()

useEffect(()=>{

const controller = new AbortController()

setLoading(true)

fetch(url, {signal: controller, signal})

.then(setData)

.catch(setError)

.finally(()=> setLoading(false))

Return ()=>{

Controller.abort()

}

}, [url])

Return {loading, data, error}

}

**React Routers**

Installing react router

-npm I react-router-dom

In index.js

Import {BrowserRouter} from ‘react-router-dom’

Root.render(

<BrowserRouter>

<App />

</BrowserRouter>

)

Import {Link,Route, Routes} from “react-router-dom”

Import {Home} from ‘.pages/Home’

Import {BookList} from ‘.pages/BookList’

Import {Book} from ‘.pages/Book’

Import {NewBook} from ‘.pages/NewBook’

Import {NotFound} from ‘.pages/Notfound’

Function App(){

Const location = useLocation();

Console.log(location)

Return (

<nav>

<Link to=”/”>Home</Link>

<Link to=”/books”>Books</Link>

<Link to=”/” replace>Anything</Link>

//replace changes the link history. Completely remove the page u are in from the history

//reloadDocument- reload the entire document

//state={‘hi’} , state management

//instead of links we can use NavLink which has className and style property

//end property used for displaying the exact active link

</nav>

<Routes>

<Route path=”/” element={<Home />}/>

<Route path=’/books’>

<Route index element={<BookList />} />

<Route path=”:id” element={<Book />}/>

<Route path=”new” element={<NewBook/>}

</Route>

//<Route path=”/books” element={<BookList />} />

//<Route path=”/books/:id” element={<Book />}/>

//<Route path=”/books/new” element={<NewBook />} />

<Route path=”\*” element={<NotFound />} />

</Routes>

)

}

Export default App;

Instead of JSX we could use js with custom hook

Let element = useRoutes([

{

Path: ‘/’,

Element: <Home />

},

{

Path: “\*”,

Element: <NotFound />

}

])

Return(

{element}

)

NotFound.js

Import {useEffect} from ‘react’

Import {useNavigate} from ‘react-router-dom’

Export function NotFound(){

Const navigate= useNavigate()

useEffect(() => {

setTimeout(()=>{

navigate(“/”)

// navigate(“/”,{state:”Error Not Page”})

navigate(-1)//go backward by 1 page

},1000)

},[])

Return <h1>NotFound</h1>

}

BookLayout.js

Import {Link, Outlet} from ‘react-router-dom’

Export function BookLayout(){

Return(

<>

<Link to=’/book/1’>Book 1</Link>

<Link to=’/books/2’>Book 2</Link>

<link to=’/books/new’>New Book</Link>

<Outlet context={{hello: “world”}}

</>

)

}

//with useSearchParams hook

Import {Link, Outlet, useSearchParams} from ‘react-router-dom’

Export function BookLayout(){

Const [searchParams, setSearchParams] = useSearchParams({n:3})

Const number= searchParams.get(“n”)

Return(

<>

<Link to=’/book/1’>Book 1</Link>

<Link to=’/books/2’>Book 2</Link>

<link to=’/books/${number}’>Book {number}</Link>

<Outlet context={{hello: “world”}}

<input

Type=”number”

Value={number}

onChange={e => setSearchParams({n: e.target.value})}

/>

</>

)

}

Book.js

Import {useParams} from ‘react-router-dom’

Export function Book(){

Const {id} =useParams();

Const obj = useOutletContext

}

Instead of browser router we can use HashRouter. Useful for shared server.

Import {HashRouter} from ‘react-router-dom’

<HashRouter>

<App/>

</HashRouter>

Instead of browser router we can use HashRouter. Useful for shared server.

Import {HashRouter} from ‘react-router-dom’

<HashRouter>

<App/>

</HashRouter>

Instead of browser router we can use HistoryRouter. Starts listening for location changes and calls the given callback with an Update when it does.

Import {unstable\_HistoryRouter} from ‘react-router-dom’

Instead of browser router we can use MemoryRouter. Useful for shared server.

Import {MemoryRouter} from ‘react-router-dom’

<MemoryRouter>

<App/>

</MemoryRouter>

Instead of browser router we can use StaticRouter. Useful for server side rendering

Import {StaticRouter} from ‘react-router-dom/server’

<StaticRouter location=’/’>

<App/>

</StaticRouter>

Instead of browser router we can use NativeRouter. Need to install React native library.

Import {StaticRouter} from ‘react-router-dom/server’

<NativeRouter location=’/’>

<App/>

</NativeRouter>

**React Query**

**Installation**

npm i @tanstack/react-query

npm run dev

import {QueryClient, QueryClinetProvider} from “@tanstack/react-query”

import {ReactQueryDevtools} from “@tanstack/react-query-devtools”

inside app.js and outside the root

const queryClient= new QueryClient()

wrap the <App/> inside the <QueryClientProvider client={queryClient}>

below <App/>

<ReactQueryDevtools>

app.js

import {useQuery, useMutation} from “@tanstack/react-query”

const posts =[

{id: 1, title: “Post 1”},

{id: 2, title: “Post 2”}

]

// /posts -> [“posts”]

// /posts/1 -> [“posts”, posts.id]

// /posts?authorId=1 -> [“posts”, {authorId: 1}]

// /posts/2/comments -> [“posts”,post.id,”comments”]

Function App(){

Const postsQuery = useQuery({

queryKey:[“posts”],

queryFn: () => wait(1000).then(()=> […POSTS])

//queryFn: () => Promise.reject(“Error Message”),

})

Const newPostMutation =useMutation({

mutationFn: title => {

return wait(1000).then(()=>

POSTS.push({id: crypto.randomUUIB(), title})

)

},

onSuccess: () => {

queryClient.invalidateQueries([“posts”])

}

})

If(postsQuery.isLoading) return <h1>Loading…</h1>

If(postsQuery.isError) {

Return <pre>{JSON.stringify(postQuery.error)}</pre>

}

Return

<div>

<h1>TackStack Query</h1>

{postsQuery.data.map(post=>(

<div key={post.id}>{post.title}</div>

))}

<button

disabled ={newPostMutation.isLoading}

onclick={()=> newPostMutation.mutate(“New Post”)}>

Add New

</button>

</div>

}

Function wait(duration){

Return new Promise(resolve => setTimeout(resolve, duration))

}

Export default App

On mutate function

Export function CreatePost(){

Const titleRef= useRef()

Const bodyRef = useRef()

Const queryClient = useQueryClient()

Const createPostMutation = useMutation({

mutationFn: createPost,

retry:3,//to refetch 3 times if response is failure, generally not used in mutation

//onSuccess: (data,variables, context) => {

//console.log(context)

//},

onSuccess: data => {

queryCLient.setQueryData([“posts”, data.id], data)

queryClient.invalidateQueries([“posts”], {exact: true})

setCurrentPage(<Post id={data.id />})

}

onMutate: variables => {//this runs first before the mutationFn

return {hi : “Bye”}

},

})

}

Pagination

**Java Script Nuggets**

**Map**

Const getAges=(person)=> person.age \* 2;

Const ages=people.map(getAges);

Console.log(ages);

Const ages = people.map(getAges);

Console.log(ages)

Const newPeople = people.map((item)=> {

Return {

firstName: item.name.toUpperCase(),

oldAge: item.age+20,

}

})

Console.log(newPeople)

**Unique values**

Const categories=[‘all’, …new Set(menu.map((item)=> item.category))];

Console.log(catgories);//set is used for unique values

Const result=document.querySelector(‘.result’);

Result.innerHTML=categories.map((category)=>{

Return `<button>category</button>`

}).join(‘’);//.join() used to remove the comma coming in between

**Dynamic object keys**

Const keyname=’computer’;

Const app={

[appState]:true

}  
app[keyname]=’apple’

Cosnt state={

Loading:true,

Name:’’,

Job:’’

}

Fucntion updateState(key,value){

State[key]=value

}

updateState(‘name’,’john’)

console.log(state)

**Filter and find**

Const people=[

{name:’bob’, age: 20, position:’developer’,salary:100},

{name:’peter’, age: 25, position:’designer’,salary:300},

{name:’sussy’, age: 30, position:’the boss’,salary:400},

{name:’anna’, age: 35, position:’intern’,salary:10}

]

Const youngPeople=people.filter((object)=>{

Return object

})

Const youngPeople=people.filter((person)=>{

Return person.age<30

})

Console.log(youngPeople[0])//need to access with index if u need a specific one even if one instance is returned where as find only return one instance

Return array of the condition passing objects

Const youngpeople=people.find((person)=>{

Return object

})

Const peter=people.find((people)=>{

Return people.name===’peter’

})

Console.log(peter)//returns an object of peter

**Reduce basics**

Const staff=[

{name:’bob’, age: 20, position:’developer’,salary:100},

{name:’peter’, age: 25, position:’designer’,salary:300},

{name:’sussy’, age: 30, position:’the boss’,salary:400},

{name:’anna’, age: 35, position:’intern’,salary:10}

]

.reduce((cummulator,instance)=>{callbackfunciton which should return the total},initial value)

const dailyTotal=staff.reduce((total,person)=>{

total+=person.salary

return total

},0)

Const cart = [

{

Title:’samsung galaxy S7’,

Price: 599.99,

Amount:1,

},

{

Title:’google pixel’,

Price: 499.99,

Amount:2,

}

]

//Let total = cart.reduce(

Let {totalItems, cartTotal}=cart.reduce(

(total, cartItem)=>{

Console.log(cartItem);

Const {amount,price}=cartItem;

Total.amount+=amount;

Total.cartTotal+=amount\*price;

Return total;

},

{

totalItems: 0,

cartTotal: 0,

}

)

cartTotal=parseFloat(cartTotal.toFixed(2))

console.log(totalItems, cartTotal);

//Console.log(total)

Const url =’https://api.github.com/users/john-smilga/repos?per\_page=100’

Const fetchRepos = async() => {

Const response = await fetch(url)

Const data= await response.json()

//Console.log(data)

Const newData= data.reduce((total,repo)=>{

Const {language}=repo;

/\*

If(language){

If(total[language]){

Total[language]=total[language]+1;

}

Else{

total[language]=1

}

\*/

Total[language]=total[language]+1 || 1

Return total;

},{})

}

Console.log(newData)

}

fetchRepos()

**Destructuring (Array)**

Condt fruits =[‘orange’, ‘banana’,’lemon’]

Const friends=[‘john’,’peter’,’bob’,’anna’,’kelly’]

Const fruit1 = fruits[0]

Const fruit2 = fruits[1]

Const fruit3= fruits[2]

Console.log(fruit1, fruit2,fruit3);

Const [john, peter, bob, anna]=friends;//just aliases need to be given variables would take the value according to the origin order, if a value not required just leave the aliases with a blank

Console.log(john, peter, bob, anna);

Let first = ‘bob’

Let second = ‘john’

[second, first]=[first, second]

Console.log(first, second)

**Destrucutring(objects)**

Const bob={

First: ‘bob’,

Last:’sanders’,

City: ‘chicago’

Siblings:{

Sister: ‘jane’

},

}

//Const firstName= bob.first;

//Const lastName= bob.last;

//Const sister = bob.siblings.sister;

Const {last: shakeAndBlake,first,city,zip}=bob;//the keys should be same

Const { last,first,city,zip, siblings:{sister: favsib} }=bob;

Console.log(first, last, city,zip,favsib);

//Console.log(firstName, lastName, sister);

In function parameter we can use destructuring

**Rest Operator**

Gather/collects the items

Destructuring/ fucntions

Rest is when we declare function, while spread while spread is when we invoke the function

Const fruits =[‘apple’, ‘orange’,’lemon’,’banana’]

Const [first, …rest]=fruits//return array

Console.log(first,rest)//apple [‘orange’,’lemon’,’banana’]

Const person={name:’john’,lastName:’smith’, job: ‘developer’};

Const {name,…rest}= person;//returns object

Console.log(name,rest);

Const getAverage=(name, ..scores)=>{

Console.log(name);

Console.log(scores);

Const average= scores.reduce((total,score)=>{

Total+=score

Return total

},0)/scores.length

Console.log(average)

}

getAverage(person.name,89,76,81,100)

**Spread Operator**

Allows and iterable to spread/expand individually inside receiver.

Split into single items and copy them

Const udemy =’udemy’;

Const letters=[...udemy];

Console.log(letters);

Const boys=[‘john’, ‘peter’, ‘bob’]

Const girls=[‘susan’,’anna’];

Const bestFriend =’arnold’;

Cosnt friends=[…boys,…girls, bestFriend];

//the value are referenced when copied rather than creating a new copy

Const newFriends =friends;

newFriends[0]=’nancy’;

console.log(friends);

console.log(newFriends);

//in objects the new variable have a copy rather than reference

const person={name:’john’, job:’developer’}

const newPerson={…person, city:’chicago’,name:’peter’}

Const person={name:’john’, job:’developer’};

**Array.from()**

Returns Array Object from an object

With a length property or an iterable object

Turns array-like/ish into array – string, nodeList, Set

Const udemy=’udemy’

Const text = document.querySelectorAll(‘.text’)

//converting node list to array

Const newText = Array.from(text).find((item)=> item.textContent ===’person’)

//Console.log(newText)

Const items = Array.from({length: 120}, (\_,index)=>{

Return index

})

Console.log(items)

Const itemsPerPage = 14

Const pages = Math.ceil(items.length / itemsPerPage)

Console.log(pages)

Const newItems = Array.from({length: pages},(\_,index)=>{

Const start = index \* itemPerPage;

Const tempItems = items.slice(start, start+itemsPerPage)

Return tempItems

})

//Console.log(pages)

Console.log(newItems)

**Optional Chaining**

Const people =[

{

Name: ‘bob’,

Location:{street: ‘123 main street’,

Timezone: {offset: ‘+7:00’}

},

{name: ‘peter’, location:{street: 123 Pine street’}},

{

Name:’susan’,

Location:{street: ‘123 ’}

},

]

People.forEach((person) => {

Console.log(person.name)

Console.log(person.location && person.location.timezone && person.location.timezone.offset)

Console.log(person?.location?.timezone?.offset || ‘hello world’)//optional chaining

}

**Callback**

Function makeUppercase(value){  
console.log(value.toUpperCase())

}

Function reverseString(value){

Console.log(value.split(‘’).reverse().join(‘’))

}

Function handleName(name,cb){

Const fullName=`${name} smith`

cb(fullName)

}

//handleName(‘peter’, makeUppercase)

//handleName(‘peter’, reverseString)

//handleName(‘susan’, function (value) {

//console.log(value)

//})

handleName(‘susan’, (value)=> console.log(value))

//array methods. setTimeout, event listeners etc

Const btn = document.querySelector(‘.btn’)

Btn.addEventListener(‘click’, function(){

Console.log();

})

**Callback Hell**

Const first = document.querySelector(‘.first’)

Const second = document.querySelector(‘.second’)

Const third = document.querySelector(‘.third’)

Const btn = document.querySelector(‘.btn’)

Btn.addEventListener(‘click’, () => {

setTimeout(() => {

first.style.color = ‘red’

setTimeout(() => {

second.style.color = ‘blue’

setTimeout( ()=> {

third.style.color = ‘green’

}, 2000)

},3000)

},5000)

})

**Promise**

Async await

Consume/use promises

Like an ice milkshake order process

Pending, rejected, fulfilled

Once rejected cannot go back to pending

Const value=2

Const promise = new Promise((resolve , reject) => {

Const random = Math.floor(Math.random() \* 3)

If(random === value){

Resolve(‘you guessed correctly’)

}

Else{

Reject(‘wrong number’)

}

//Resolve(‘Hello world’)

//Reject(‘there was an error’)

})

Console.log(promise)

Promise.then((data) => console.log(data)).catch((err)=> console.log(err))

**Promise Example**

Const btn = document.querySelector(‘.btn’)

Btn.addEventListener(‘click’, () => {

addColor(1000,’.first’,’red’)

.then(() => addColor(3000,’.second’,’blue’))

.then(() => addColor(2000,’.third’,’green’))

.catch((err)=>console.log(err))

})

Function addColor(time, selector, color){

Const element = document.querySelector(selector)

Return new Promise((resolve,reject)=> {

If(element){

setTimeOut(()=>{

element.style.color = color

resolve(data)

}, time)

} else{

Reject(‘there is no such element: ${selector}’)

}

})

}

**Async/await**

Async must be present, always returns a promise

Await waits till promise is settled

Error handling- try/catch block

//Const example = async() =>{

//Return ‘hello there’// async function aways return promise

//}

//Async function someFunc (){

//Const result = await example()

//Console.log(result)

//}

////Console.log(example())

//SomeFunc()

Const users = [

{id: 1, name: ‘john’},

{id: 2, name: ‘susan’},

{id: 3, name: ‘anna’},

]

Const articles =[

{userId: 1, articles : [‘one’,’two’,’three’]},

{userId: 2, articles : [‘four’,’five’]},

{userId: 3, articles: [‘six’,’seven’,’eight’,’nine’]},

]

Const getData = async () =>{

Const user = await getUser(‘john’)

Console.log(user)

}

Const getData= async() =>{

Try{

Const user= await getUser(‘john’);

Const articles= await getArticles(user.id)

Console.log(user)

//If(user){

//Const articles = await getArticles(user.id)

//Console.log(articles)

//}

} catch(error){

Console.log(error)

}

}

getData()

//getUser(‘susans’)

//.then((user) => getArticles(user.id))

//.then((articles) => console.log(articles))

//.catch((err) => console.log(err))

Function getUser(name){

Return new promise((resolve, reject) => {

const user = users.find((user)=> user.name === name)

if(user){

return resolve(user)

} else{

Reject(‘No such user with name: ${name }’)

}

})

}

Function getArticles(userId){

Return new Promise ((resolve, reject) => {

Const userArticles =articles.find((user) => user.userId === userId)

If(userArticles){

Return resolve(userArticles.articles)

}

Else{

Reject(‘wrong ID’)

}

})

}

**Fetch**

Fetch API- Browser API for HTTP(AJAX) Requests

GET Requests, support other methods as well

Returns promise

Const url =’https://www.course-api.com/react-tours-projects’

//Fetch(url)

//.then((resp) => resp.json())

//.then((data) => console.log(data))

//.catch((err) => console.log(err))

Const getTours = async ()=>{

Try{

Const resp = await fetch(url)

Consr data = await resp.json()

Console.log(resp)

Return data

} Catch(error){

Console.log(“error”)

}

Console.log(getTours().then())

**Fetch errors**

Const url =’https://www.course-api.com/react-tours-projects’

Const getTours = async ()=>{

Try{

Const resp = await fetch(url)

If(!resp.ok)

{

Consr msg =`There was an error “${resp.status} ${resp.statusText}”`

Throw new Error(msg)

}

Consr tours = await resp.json()

Console.log(tours)

Return data

} Catch(error){

Console.log(“error”)

}

Console.log(getTours().then())

**Width/Height**

Console.log(‘Height’, window.innerHeight)//gives the dimensions of the window/screen

Console.log(‘Width’, window.innerWidth)

Const btn = document.querySelector(‘.btn’)

Const box = document.querySelector(‘.box’)

Btn.addEventListener(‘click’, () => {

Const dimensions = box.getBoundingClientRect()//gives the dimensions of the element selected

Console.log(dimensions)

})

**Timestamp**

Console.log(new Date());//todays date Sun June 1 2024 08:42:12 GMT-0600 (pacific Daylight Time)

//Unix Time

// January 1, 1970

//1s=1000ms

Console.log(Date.now());

Console.log(new Date().getTime());

Console.log(new Date().valueOf());// all three same result

setTimeout(()=> {

console.log(Date.now())

}, 1000)

Let people =[]

People =[…people, {id: Date.now(), name:’peter’}];

setTimeout(()=> {

people =[..people, {id: Date.now(), name: ‘peter’}]

}, 1000)

Console.log(new Date(constant value))

Const now = Date.now()

Const futureDate = new Date(now + 1000 \* 60)

Console.log(futureDate)

Console.log(new Date())

//difference between date

Const firstDate = new Date();

Const secondDate = new Date(2021, 8, 28);

Const firstValue= firstDate.getTime();

Const secondValue= secondDate.getTime();

Console.log(firstValue);

Console.log(secondVlaue);

Const timeDifference= secondValue-firstValue;

Console.log(timeDifference);

Const minutes = timeDifference / (1000\* 60)

Console.log(minutes)

Const hours = timeDifference / (1000 \* 60 \* 60);

Console.log(hours)

//cookies example

Res.cookies(‘token’, token, {

httpOnly: true,

expires: new Date(Date.now() + oneDay),

secure: process.env.NODE\_ENV === ‘production’,

signed: true,

});

**ECMAScript 2022**

To make this work we have to make the script type module

at() – takes integer and returns the items at that index – string, array

const scores =[99,90,100]

const oldLast = scores[scores.length -1]

console.log(oldLast)

const newLast = scores.at(-1);

console.log(newLast)

const channel = ‘Coding Addict’;

console.log(channel.at(0))

//fetch(‘https://www.course-api.com/react-tabs-project’)

//.then((resp) => resp.json())

//.then((data) => console.log(data))

//can use the try catch block

Const fetchData = async () =>{

Const resp = wait fetch(‘https://www.course-api.com/react-tabs-project’)

Const data = await resp.json()

Console.log(data)

}

fetchData();

if another module already present

const fetchTabs = () => {

return fetch(‘https://www.course-api.com/react-tabs-project’)

.then((resp) =>

Resp.json()

)

}

Export default fetchTabs

Top level Await – Basic Example

Import fetchTabs from ‘./fetchTabs.js’

Const tabs =await fetchTabs();

Console.log(tabs)

**Get Element Helper**

Const heading = document.querySelector(‘.heading’);

Const listItems = document.querySelectorAll(‘.list-item’);

Const getElement = (selector, isList) => {

//If(isList){

//Const el = […document.querySelectorAll(selector)]

//If(el.length < 1){

//Throw new Error(‘Please double check selector: ${selector}’)

//}

//Return el;

//}

//Const el = document.querySelector(selector);

//If (el) return el;

Const el = isList

?[…document.querySelectorAll(selector)]

: document.querySelector(selector);

//not a list – exist or not

//list – empty or not

//If(!isList && el) return el;

//If(isList && !el.length < 1) return el;

If((!isList && el)|| (isList && !el.length <1)) return el;

Throw new Error(`Please double check selector : ${selector}`);

};

//Console.log(getElement(‘.heading’))

Console.log(getElement(‘.list-ite’, true))

//“For in” loop – iterate over object properties

//not advised to use it on arrays, especially if the order is important

// on arrays use ‘for of’ loop instead

Const person ={

Name: ‘john’,

Age:’25’,

Status: ‘student’

}

For(const propertyName in person){

Console.log(`${propertyName} : ${person[propertyName]}`));

}

Three methods to convert objects into arrays

Object.keys() – converts property names into array

Object.values() – convert property values into array

Object.entries() – convert both

Condt person ={

Name: ‘john’,

Age:25,

Status: ‘student’

}

Const result = Object.entries(person);

Console.log(result);

Const newResult = result.map((item) => {

Const [first,second] = item;

Console.log(first, second)

Return first;

})

For(const [first,second] of result){

//For(const item of result){

//Const [first, second] =item;

Console.log(first, second);

}

**Set()-set object**

-Stores a collection of unique values of any type

New Set()

add(value)

delete(value)

clear()

has(value)

//iterators

Entries(), keys(), values(), forEach()

Const unique = new Set()

Unique.add(‘first’)

Unique.add(‘second’)

Unique.add(‘third’)

Unique.add(‘first’)//ignored

Const result = unique.delete(‘five’);

Console.log(result);

Const result2 = unique.clear();

Console.log(result2);

Const isValue = unique.has(4)

Console.log(isValue);

Console.log(unique)

Const product =[

{

Title: ‘high-back bench’,

Company: ‘ikea’,

},

{

Title:’albany table’

Company: ‘marcos’

},

{

Title: ‘accent chair’,

Company:’caressa’,

},

{

Title:’wooden table’,

Company:’ikea’,

},

];

//Const companies = product.map((item) => item.company)

//Const uniqueCompanies = new Set(companies)

//Console.log(uniqueCompanies)

//Const finalCompanies = [‘all’, …uniqueCompanies];

//Console.log(finalCompanies);

Const result =[‘all’, new Set(product.map((item)=> item.company))];

Console.log(result)

**String includes()**

-checks if a string contains another string

Const product =[

{

Title: ‘high-back bench’,

Company: ‘ikea’,

},

{

Title:’albany table’

Company: ‘marcos’

},

{

Title: ‘accent chair’,

Company:’caressa’,

},

{

Title:’wooden table’,

Company:’ikea’,

},

];

Const text=’a’

Const filteredProducts = products.filter((product)=>

Product.title.toLowerCase().includes(text)

)

Console.log(filteredProducts)

//Const firstName =’john’

//Const result = firstName.includes(‘j’,1)

//Const product ={

//Title: ‘Leather Chair’

//};

//Const result = product.title.includes(‘le’)

//Console.log(result)

**Array includes()**

-checks if the item is in an array

Useful in the conditional statement

Const groceries =[‘milk’,’bread’,’meat’]

Let randomItem = ‘lemon’

//const isIncluded = groceries.includes(ranodmItem, 1)

//console.log(isIncluded)

If(groceries.includes(randomItem)){

Console.log(‘Yeah! It’s on the list’)

}

**FormData API**

Const form = document.querySelector(‘.form’)

Form.addEventListener(‘submit’, (e)=>{

e.preventDefault();

//spread out – entries, values, keys

Const formData = new FormData(e.currentTarget);

//check for value

Const name=formData.get(‘name’)

//append

formData.append(‘test’,’testValue’);

//delete

//formData.delete(‘test’)

formData.delete(‘email’)

const checkName= formData.has(‘name’)

console.log(checkName);

Const entries =[…formData.entries()]

Console.log(entries)

//Const values =[…formData.values()]

//Console.log(entries)

//Const keys =[…formData.keys()]

//Console.log(entries)

For(let [name,value] of formData){

Console.log(`${name}=$values`);

}

})

Const form = document.querySelector(‘.form’)

Form.addEventListener(‘submit’, (e)=>{

e.preventDefault();

Const formData = new FormData(e.currentTarget);

Const values= […formData.values()];

If(values.includes(‘ ’)){

Console.log(‘please enter all objects’)

Return;

}

Const formObject = Object.fromEntries(formData);

Console.log(formObject);

e.currentTarget.reset();

**Debounce**

Const btn = document.querySelector(‘.btn’);

Const debounce =() => {

Const timeoutID = setTimeout( () =>{

Console.log(‘you clicked me’);

},2000);

Console.log(timeoutID)

clearTimeout(timeoutID)

console.log();

}

//Btn.addEventListener(‘click’, ()=> {  
//console.log(‘you clicked me’)

//})

Btn.addEventListener(‘click’, debounce)

//delay logic

//so it runs 2s after last click

//setTimeout returns id, which pass into clearTimeout

//new code

Const btn = document.querySelector(‘.btn’)

Const debounce = () => {

Let timeoutID;

Return () => {

Console.log(timeoutID);

clearTimeout(timeoutID);

timeoutID = setTimeout(() => {

console.log(‘you clicked me’)

},2000)

}

}

Btn.addEventListener(‘click’, debounce())

**Websockets**

Websockets

1. Full-duplex bi-directional communication
2. WebSocket is a HTTP upgrade
3. Easy to implement and standardized
4. Only sends header once

Can I use .com

Polling & Long Polling

-Alternative to WebSockets

Much better backwards compatibility

-Polling

Send AJAX request every X amount of seconds for new data(not true real time)

-Long Polling

Send request to server and keep connection open until new data

Server Sent Events

Another ‘real-time’ alternative

Uses EventSource API to send messages from server

Not truly requires an event loop

Generally, requires an event loop

No binary message capability

Intended Use Case

WebSockets not == replacement for HTTP

WS is an upgrade for HTTP

HTTP provides automatic caching!

WS often needs special configuration of load balancing

Can’t communicate with REST

Use when you need full-duplex connection

Useful for web-based games, chatting applications, anything which needs low-latency real-time connection!

WebSocket Clients

Used to interface with WebSocket Server

Built in many languages! (including Python!)

Clients exist for MicroPy and Arduino!(IoT)

Most common clients is web based and uses Javascript

Require the Server to be able to interface WS

WebSocket Clientside Code

Const socket = new WebSocket(‘ws://localhost:8000’)

Socket.onopen=(event)=>{

Socket.send(‘PyCon AU!!’)

});

Socket.onmessage=(event)=>{

Console.log(event.data);

});

SocketIO

Javascript library for manipulating Websockets

Include fallback mechanism and auto reconnection

Handles disconnection and connection events

Namespacing and Room broadcasting

SocketIO Clientside Code

Var socket =io(‘http://localhost:8000/<MY\_NAMESPACE>’)

Socket.on(‘connect’),()=>{

Socket.emit(‘event\_on\_my\_server’,data=”PyConAU!”);

});

Socket.on(‘my\_custom\_event’,(data)=>{

})

Flask

Socket=Sockets(app)

@socket.route(‘/my\_sockets’)

Def my\_socket\_event(ws):

While not ws.closed:

Message=ws.recieve();

Ws.send(message)

Performce comparision

http and WebSocket have the same sized header

2byte/msg overhead

Socketio increases latency and intial connection under the hood starts:

Uses AJAX Long Olling initially and then upgrades

Deployment

Even let and Gevenet ti monkey patch async

Or just use standard threading

Async web framework are ideal

Use message queue to run multiple instances behind, load balancers with sessions workers

**SEO(Search Engine Optimization)**

Benefits of SEO

1. Unlike paying for ads, search traffic is free.

2. Organic traffic is typically consistent once you’re ranking high

3. You have the opportunity to reach massive audiences.

How google works

* Crawling and indexation

The crawlers crawl through the seeds(most known websites) and their links to other pages to crawl the numerous pages. Then bring the info back to the server and index according to the content and findings.

* Google ranking algorithm

Factors:

Backlinks

Search Intent

Content depth

Keyword Research

How do you choose keywords worth targeting

-check if you keyword has search demand

Search demand represent the volume of monthly searches made for a keyword

-check the traffic potential of the topic

Traffic potential represent the total search traffic you could get if you were to rank at the top of Google for you keyword

-Assess the business potential of the keyword or topic

For golf website selling golf clubs

Ranking of business potential is:

3. buy used golf clubs

2. best golf clubs

1. what is a handicap in golf club

- see if you can match searcher intent

Search intent is the reason behind the searcher’s query.

Match search engine by-

Showing the search engines that your page will fulfill their goal

To deliver the most relevant results for any query.

Use the 3c’s of search intent.

Content type: Blog posts, videos, product pages, category pages, landing pages

Content format for blogs

How-to-guide

Step-by-step tutorials

List posts

Opinion editorials

Content format for Landing pages

Tool

Calculator

Content angle

Its basically your ‘hook’ as to why someone should click and visit your page

For query:- how to swing golf ball

The top results were vid and instructions one

And mentioning the word basic is also impactful for the searchers

-Determine whether you can rank for your keyword

Keyword research

The process of finding keywords that people are searching for in search engines.

1. generate keyword ideas

2. validate whether those keywords are worth going after

Keyword Research Tools

Show you information on keywords

-search volume

-keyword difficulty scores

-other seo metrics

They should help you discover potential topics worth going after

Keyword generator of ahrefs

For a golf website where u provide link for other buy pages, where u get commission if purchase made via our website

1. come up with a list of seed keywords

Seed keyword is a broad keyword related to you niche

After choosing, here golf balls, golf clubs, golf hats

Checking the 5 points for choosing the keyword

1. we want keywords that have search demand

Filter the search volume by selecting the volume filter

2.keywords worth traffic potential

Then u can check the serp

3.keywords with business potential

Business potential on the keyword monthly 16k search is great

4. we need to be able to match search intent

Search intent previous was shopper intent so we changed our keyword to ‘best golf club’

Which 5k search and more similar to matching search intent

Keyword modifier

An add on to a base keyword

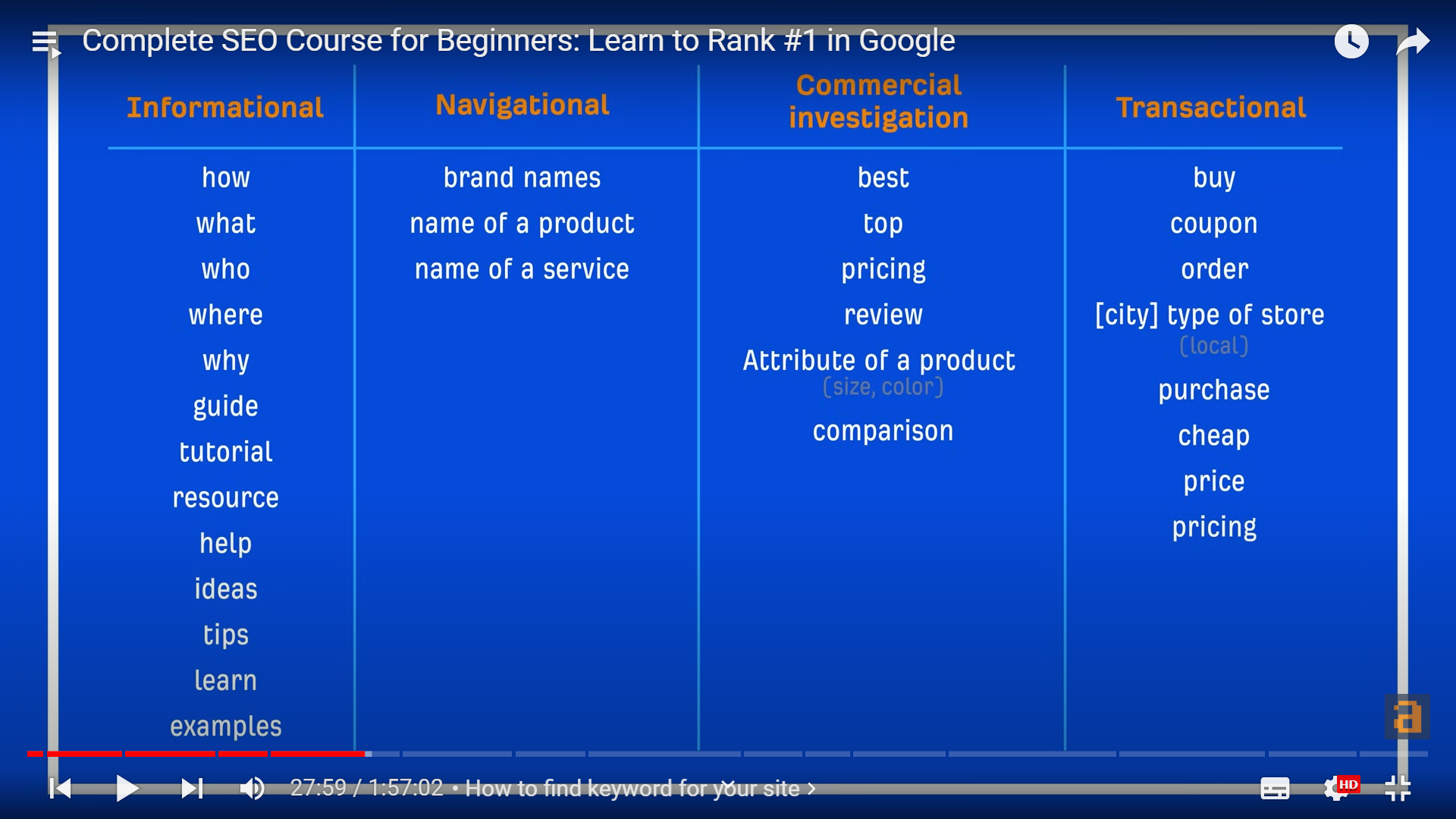
Best, top , current year to the main keyword

By using this to our favor:

Keywords that likely have business potential

Keywords where we can match searcher intent

Best top vs review



The best way to find these keywords is to look at pages that drive the most search traffic to your competitors’ sites.

Competitor = business competitor

Competitor = search competitor

Go to the top pages and select the domain and select the top pages

To find which pages bring the most traffic

Exhaust the list of competitors until you are satisfied.

Websites that rank for keywords that you’d want to rank for.

5.we want to know how hard it’ll be to rank at the top of Google for that keyword

When it comes to ranking in google, you need to understand who you’ll be up against before you target a keyword

Competitors(SEO)

Pages and websites that rank at the top of Google for your target keywords.

3 main things to consider:

Search intent

Metrics of the top-ranking pages and websites

Topical authority of the top-ranking websites.

In general, if the top pages include the primary keyword or a vacation of it in the title and/or URL, they’re likely targeting that keyword.

Metrics of the top-ranking pages and websites

-the number of websites that are linking to the page (referring domains)

Can I get more quality backlinks than the top-ranking pages?

-website authority

Domain rating- represent the overall strength of a website’s backlink profile.

Is my website in a similar DR range or higher than the top-ranking websites.

Topical authority of the top-ranking websites

Google wants to rank pages from authoritative sources

Is my website equally or more topically authoritative than the top-ranking websites

Ranking Difficulty