1. Scroll Suave

Ao clique no menu, a pagina scrolla até ao titulo que o href faz referencia

Const chamando construtor class enviando como parâmetro a lista de links e em seguida utilizando o init para dar inicio a função construtora

*const* scrollsuave  = new ScrollSuave('[data-menu="suave"] a[href^="#"]');

scrollsuave.init();

Poderia ser passado aqui, as opções de scroll como um objeto;

*const* options = {

  behavior: 'smooth',

  block: 'start,'

}

O Codigo de scroll suave é uma constructor class que recebe como constructor o link e as opções de scrollIntoView,   
Iniciada por init() verificando com o if, caso tenha sido passado o link, executa o método AddLinkEvent que irá adicionar um event listener de click para cada link da lista passada e como call-back irá executar a scrollToSection  
Para ser executado o método ScrollToSection, ele deve receber da class por bind o this.  
Após receber um novo this por bind, previne o padrão de ir direto ao link,   
seleciona o atributo do target href , e seleciona a section com o href,   
Para finalizar, executa o método ScrollIntoView em section, com parâmetro Options   
O Init deve sempre retornar um this, caso seja desejado emendar outro método na chamada da constructor class

export default *class* ScrollSuave { *// função construtora class sendo exportada*

*constructor*(*links*, *options*) { *// recebendo como parametro o link da lista, e possiveis opções opções do scroll*

    this.linksInternos = document.querySelectorAll(links); *// selecionando a lista com o parametro passado*

    if (options === undefined) { *//  caso não seja definida opções, terá pré definidas*

      this.options = { behavior: "smooth", block: "start" };

    } else {

      this.options = options;

    }

*// passando para ScrollToSection este this com bind,  para ele sempre ser executado com este this (objeto)*

*// caso não tivesse sido passado esse parametro, em addlinkevent ele teria como parametro o link, não o objeto*

    this.scrollToSection = this.scrollToSection.bind(this);

  }

  scrollToSection(*event*) { *// metodo que selecionara o link da sessão*

    event.preventDefault(); *// previnindo o padão, de ir ate o link*

*const* href = event.currentTarget.getAttribute("href"); *// selecionando o href*

*const* section = document.querySelector(href); *// const section seleciona com o href a sessão*

    section.scrollIntoView(this.options); *// section com metodo scrollIntoView com o parametro options*

  }

  addLinkEvent() { *// metodo que irá dar o evento de click a todos links*

    this.linksInternos.forEach((*link*) => {

      link.addEventListener("click", this.scrollToSection);

    });

  }

  init() { *// inicio da função*

    if (this.linksInternos.length) { *// caso seja passado link é executado*

      this.addLinkEvent();

    }

    return this; *// retornando para caso o usuario deseja-se utilizar outras funções em sequencia*

  }

}

1. Accordion

É lista que terá uma área escondida, que mostrara ao clique

Const com parâmetro da lista, e realizando o init

*const* accordion = new Accordion('[data-anime="accordion"] dt');

accordion.init();

Também é uma constructor class que recebe como parâmetro o local da lista e inicia com init  
Verifica se foi passado a lista, da um toggle no primeiro item, e executa o método de event de click  
O event click realiza um for each adicionando o click e com call-back tem uma arrow function que irá retornar a chamada da função toggleAccordion mencionando o this item,   
A função toggleAccordion realiza o toggle tanto na lista (this) tanto na próxima(sibling)

export default *class* Accordion { *// função class constructor*

*constructor*(*list*) {

    this.accordionList = document.querySelectorAll(list); *// selecionando a lista com base no parametro*

    this.activeClass = "ativo"; *// nome da classe que o item ira receber quando ativo*

  }

  toggleAccordion(*item*) { *// metodo que irá realizar a mudança (toggle)*

    item.classList.toggle(this.activeClass); *//  adiciona e remove a classe do item*

    item.nextElementSibling.classList.toggle(this.activeClass); *//  e do item de baixo*

  }

  addAccordionEvent() { *// metodo que adiciona o evento em todos itens da lista*

    this.accordionList.forEach((*item*) => {

      item.addEventListener("click", () => this.toggleAccordion(item));*// ao click, executa o metodo toggle com o item utilizando uma arrow function*

    });

  }

  init() { *// metodo inicial*

    if (this.accordionList.length) { *// se selecionado a lista \/*

      this.toggleAccordion(this.accordionList[0]); *// o primeiro item irá ativar a função*

      this.addAccordionEvent(); *// adiciona o evento de click*

    }

  }

}

1. Navegação por tab

Método em que temos um menu, e uma área de conteúdo que são de index igual, assim ao clique no item do menu, a section correspondente recebe uma class de ativo, e as outras de inativo

Const recebendo o menu e o conteúdo como parâmetro, e dando init

*const* tabNav = new TabNav('[data-tab="menu"] li', '[data-tab="content"] section')

tabNav.init()

o funcionamento da class contructor de navegação por tabs é receber o menu e o conteúdo, ativando o item de index 0  
Em seguida adicionando evento de click em cada item do menu, ao item ser clicado, ele aciona o método que remove a classe de todos os conteúdos, e em seguida adiciona a classe ativo ao conteúdo de index correspondente ao do menu clicado   
E para adicionar efeito de animação, cada contente recebe um data-set com um nome de classe (right e down) e puxamos esse valor a uma const, e quando formos adicionar a classe de ativo, também adicionamos a classe de animação com o valor na const direção

export default *class* TabNav {

*constructor*(*menu*, *content*) {

    this.tabMenu = document.querySelectorAll(menu); *// selecionando o menu*

    this.tabContent = document.querySelectorAll(content); *// e o conteudo*

    this.activeClass = "ativo"; *// classe que as classes irem receber e remover*

  }

  activeTab(*index*) {

    this.tabContent.forEach((*section*) => {

    section.classList.remove(this.activeClass); *// é removido a classe ativo de todos os itens*

    });

*const* direcao = this.tabContent[index].dataset.anime; *//adiciona a direção o valor de dataset de cada content*

    this.tabContent[index].classList.add(this.activeClass, direcao); *// e adicionado a classe ativo na section com o index referente ao parametro passado, e adiciona tambem a classe com valor da const direção*

  }

  addTabNavEvent() {

    this.tabMenu.forEach((*itemMenu*, *index*) => {*// passado com foreach o menu e seu index*

      itemMenu.addEventListener("click", () => this.activeTab(index)); *// ao clique, ativa arrowfunction com metodo de ativação com o index referente ao clique*

    });

  }

  init() {

    if (this.tabMenu.length && this.tabContent.length) { *// caso for passado o menu e a section*

      this.activeTab(0); *// o primeiro item ativa o metodo activetab*

      this.addTabNavEvent();*// e acionado o metodo de adicionar evento de click*

    }

  }

}

1. Modal

Modal é uma janela que é aberta ao clique, e fechada ao clique no X ou fora da caixa

Modal sendo iniciado passando onde irá abrir, o X de fechar, e a área atras do modal

*const* modal = new Modal('[data-modal="abrir"]','[data-modal="fechar"]','[data-modal="container"]' );

modal.init();

modal.toggleModal() *// podemos utilizar os metodos do modal fora do corpo do modal*

*//aqui por exemplo realizamos o toggle para o site iniciar com a janela aberta*

Para o funcionamento do modal, precisamos criar no HTML uma section com a área atras do modal, e o modal por cima dela   
O CSS deve ter display none, e flex quando for adicionado a classe ativo  
A class constructor deve iniciar recebendo como parâmetro o botão de abrir, o de fechar e a area do modal,  
a partir do init verificamos se foi passado os 3 parametros e acionamos o método de adicionar evento  
Os botões de abrir e fechar vão direto para o método acionador, já o do modal para um método que verifica se o clique foi dentro da caixa ou não  
Esse verificador verifica com if se o clique foi na área atras do modal, caso positivo passa para o método ativador  
O método ativador, caso o clique for fora da caixa modal ou no X ou no de abrir, previne o padrão e manda para o método de toggle  
O método de toggle realiza o toggle da classe ativo

export default *class* Modal {

*constructor*(*botaoAbrir*, *botaoFechar*, *containerModal*) {

  this.botaoAbrir = document.querySelector(botaoAbrir);

  this.botaoFechar = document.querySelector(botaoFechar);

  this.containerModal = document.querySelector(containerModal); *// area 100% atras do modal*

*// devido o callback ser ativado por outro this, passamos por bind o this objeto*

  this.eventToggleModal = this.eventToggleModal.bind(this)

  this.cliqueForaModal = this.cliqueForaModal.bind(this)

  }

  toggleModal() {

        this.containerModal.classList.toggle('ativo'); *// adiciona ou remove a classe ativo*

  }

  eventToggleModal(*event*) { *// metodo que previne o padrão, e realiza toggle*

    event.preventDefault();

    this.toggleModal()

  }

  cliqueForaModal(*event*) { *// metodo do clique fora*

    if (event.target === this.containerModal) { *// se o target do clique for na area atras do modal*

      this.eventToggleModal(event); *// ativa o metodo que ativa o toggle*

    }

  }

  addModalEvents() { *// cada botão, e a caixa recebe evento de clique e callback de toggle ou verificar clique fora*

    this.botaoAbrir.addEventListener('click', this.eventToggleModal);

    this.botaoFechar.addEventListener('click', this.eventToggleModal);

    this.containerModal.addEventListener('click', this.cliqueForaModal);

  }

  init() { *// inicio do modal*

    if(this.botaoAbrir && this.botaoFechar && this.containerModal){ *// caso seja passado os 3 parametros*

    this.addModalEvents() *// inicia o metodo de adicionar evento*

    }

    return this

  }

}

1. ToolTip

Tooltip é uma caixa de dialogo que irá aparecer quando ocorrer o evento mouse over

Para iniciar, deve haver no html uma div com a tag aria-label com o texto desejado, a classe vai ser inserida e o elemento criado via JS

*const* tooltip = new Tooltip('[data-tooltip]') *// passando a div como parametro*

tooltip.init()

export default *class* Tooltip {

*constructor*(*tooltips*) {

    this.tooltips = document.querySelectorAll(tooltips);

*// bind necessario pois ativado dentro de outros metodos*

    this.onMouseLeave = this.onMouseLeave.bind(this);

    this.onMouseOver = this.onMouseOver.bind(this);

    this.onMouseMove = this.onMouseMove.bind(this);

  }

*//função responsavel por acionar o metodo de criar a tolltip e os eventos de mouse over e leave*

  onMouseOver({ *currentTarget* }) {  *// desestruturando o event, e utilizando o current target*

    this.criarTooltipBox(currentTarget); *// acionando o metodo de criação da tooltip*

    currentTarget.addEventListener("mousemove", this.onMouseMove); *// com a div criada, adiciona o callback de mouse move e leave*

    currentTarget.addEventListener("mouseleave", this.onMouseLeave);

  }

  onMouseMove(*event*) { *// metodo que indica a posição da caixa*

    this.tooltipBox.style.top = **`**${event.pageY + 20}px**`**;

    if (event.pageX + 240 > window.innerWidth) { *// cado a posição seja muito proxima a lateral da janela,*

      this.tooltipBox.style.left = **`**${event.pageX - 195}px**`**;*// a caixa recua 195px*

    } else {

      this.tooltipBox.style.left = **`**${event.pageX + 20}px**`**; *// caso contrario distancia de 20px*

    }

  }

  onMouseLeave({ *currentTarget* }) { *// quando acionado o mouse leave, remove os eventos*

    this.tooltipBox.remove();

    currentTarget.removeEventListener("mouseleave", this.onMouseLeave);

    currentTarget.removeEventListener("mousemove", this.onMouseMove);

  }

  criarTooltipBox(*element*) {

*const* tooltipBox = document.createElement("div"); *// cria o elem. div*

*const* text = element.getAttribute("aria-label"); *// seleciona o texto passado no elemento*

    tooltipBox.classList.add("tooltip"); *// adiciona a classe tooltip*

    tooltipBox.innerText = text; *// coloca o texto*

    document.body.appendChild(tooltipBox); *// e adiciona na div, a div criada aqui*

    this.tooltipBox = tooltipBox; *// this.tooltip recebe a div criada aqui*

  }

  addTooltipsEvent() { *//*

    this.tooltips.forEach((*item*) => {

      item.addEventListener("mouseover", this.onMouseOver); *// adicionamos o evento mouseOver acionando a função onMouseOver*

    });

  }

  init() { *// aqui inicia a class*

    if (this.tooltips.length) { *// caso seja passado a tooltip*

      this.addTooltipsEvent(); *// iniciamos o metodo de adicionar evento*

    }

    return this;

  }

}

1. Anima Numeros

Uma class constructor que tem um observador, que após a sessão em questão receber uma classe, realiza o inicio da animação  
A animação trata-se de um incremento variado de 0 até o numero indicado

Em outro modulo estamos passando ao anima números os números, a sessão, e a classe a ser observada

*const* animaNumeros = new AnimaNumeros('[data-numero]','.numeros', 'ativo');

      animaNumeros.init()

    } catch (erro) {

      console.log(erro);

    }

  }

Inicia observando se foi passado os parâmetros necessários, caso sim é iniciado o método que irá observar mutações  
Caso haja, ele iniciara um novo método que irá analisar se a mutação foi a adição da classe ativo, caso sim ele remove o observador, e inicia o método de animação dos números  
então é realizado um foreach nos números acionando o método de incremento  
Este método ira receber o numero, criar um incremento com base no total, e com setinterval irá realizar a contagem até passar do total, quando ocorrer o numero volta para seu estado atual, e o contador é limpo, cada foreach é realizado com um multiplicador de setinterval randômico

export default *class* AnimaNumeros {

*constructor*(*numeros*, *observerTarget*, *observerClass*) {

    this.numeros = document.querySelectorAll(numeros); *// numeros a serem recebidos*

    this.observerTarget = document.querySelector(observerTarget); *// sessão a ser observada*

    this.observerClass = observerClass; *// classe a ser observada*

    this.handleMutation = this.handleMutation.bind(this);

  }

*// metodo estatico pertencente ao prototipo da class constructor Anima numeros*

*// é necessario ser static pois contem const nãod eclarada em constructor, mas não já necessidade de declarar pois não é utilizado fora*

*// é possivel utilizar esse metodo em qualquer numero no site atraves de: Animanumeros.incrementarNumero(~objeto que contenha numero no dom~)*

  static incrementarNumero(*numero*) { *//*

*const* total = +numero.innerText; *//recebendo como numero o innertext*

*const* incremento = Math.floor(total / 100); *// dividindo o numero por 100 e ajustando com floor*

*let* start = 0; *// inicio da contagem*

*const* timer = setInterval(() => {

      start += incremento; *//sera incrementado*

      numero.innerText = start; *// e atualizado na tela*

      if (start > total) {  *// caso o total seja maior que a contagem*

        numero.innerText = total; *// o numero recebe o total*

        clearInterval(timer); *// e limpa o setinterval*

      }

    }, 25 \* Math.random()); *//ocorre a cada 25milesimos, vezes numero aleatorio*

  }

*// por incrementar numero ser static, pertence ao prototipo de this(AnimaNumeros)*

  animaNumeros() {

    this.numeros.forEach((*numero*) => *// para cada numero aciona o metodo de incremento*

      this.constructor.incrementarNumero(numero)

    );

  }

  handleMutation(*mutation*) {*//ao ser acionado quando houver uma mutação*

    if (mutation[0].target.classList.contains(this.observerClass)) { *// se a mutação for classe ativo no target, informado em mutation[0]*

      this.observer.disconnect(); *// se sim, para de observar*

      this.animaNumeros();*// e ativa a animação*

    }

  }

  addMutationObserver() {

    this.observer = new MutationObserver(this.handleMutation); *// criando objeto predefinido 'MutationObserver' com o handlemutation como callback*

    this.observer.observe(this.observerTarget, { attributes: true }); *//utilizando na const observer o metodo observer, e passando como parametro a sessão a ser observada, e passando a opção atributos, para eles serem observados*

  }

  init() { *// inicio*

    if (this.numeros.length && this.observerTarget) {

      this.addMutationObserver(); *// caso recebido, inicia o observador*

    }

    return this;

  }

}

1. Fetch animais – Criando html com json

Realizamos fetch de um arquivo json e criamos div no html para alocar essas informações  
Essa função vai ser implementada com function, pois utilizamos await

fetchAnimais('../../animaisapi.json', '.numeros-grid');

iniciando o método de fetch, primeiro é iniciado a função criar animais, onde é realizado o fetch com await e para cada objeto do json é passado pela function de preencherAnimais, que ira ativar a função de createAnimal, que ira retornar o html pronto e pela preencher animal sera inserida no html

A animação é inserida como declaração no Criar animais, assim já está sendo executado o observer, e quando houver a mudanã sera executada

import AnimaNumeros from "./anima-numeros.js";

export default *function* fetchAnimais(*url*, *target*) {

*const* numerosGrid = document.querySelector(target); *// selecionando em numeros grid a sessão passada por parametro*

*function* createAnimal(*animal*) { *// funçao que cria a div e aloca as informações puxadas do json*

*const* div = document.createElement("div");  *// cria a div*

    div.classList.add("numero-animal"); *// adiciona a classe na div*

    div.innerHTML = **`**<h3>${animal.especie}</h3><span data-numero>${animal.total}</span>**`**; *// adiciona dentro da div as informações do json*

    return div; *// retorna a div para ser inserida no html*

  }

*function* preencherAnimais(*animal*) {

*const* divAnimal = createAnimal(animal); *// ativa a função de criação da div com as informações*

    numerosGrid.appendChild(divAnimal); *// adiciona a div no html*

  }

*function* animaAnimaisNumeros() { *// função externa de animação dos numeros*

*const* animaNumeros = new AnimaNumeros("[data-numero]", ".numeros", "ativo");

    animaNumeros.init();

  }

*//3*

  async *function* criarAnimais() { *//função que realiza o fetch do json*

    try {

*const* animaisResponse = await fetch(url); *// faz o fetch e aguarda receber*

*const* animaisJSON = await animaisResponse.json(); *// transforma em json*

      animaisJSON.forEach((*animal*) => {

        preencherAnimais(animal);*// para cada objeto do json realiza função preencher*

      });

      animaAnimaisNumeros(); *// é executado a função de animação*

    } catch (erro) {

      console.log(erro);

    }

  }

  return criarAnimais();

}

1. Fetch bitcoin

Realizamos um fetch de json utilizando then, e colocamos o valor recebido e tratado dentro do html

fetchBitcoin('https://blockchain.info/ticker', '.btc-preco');

export default *function* fetchBitcoin(*url*, *target*) { *//recebendo como parametro o url do fetch, e onde sera colocado*

  fetch(url) *// recebendo o fetch*

    .then(*response* => response.json()) *//transformando em json*

    .then((*bitcoin*) => {

*const* btcPreco = document.querySelector(target); *// selecionando a section*

      btcPreco.innerText = (1000 / bitcoin.BRL.sell).toFixed(4); *// colocando o preço tratado na section*

    }).catch(*erro* => console.log(Error(erro)));

1. Anima Scroll

Função que coleta a posição de cada sessão do site, e ao scroll chegar na posição, realiza a animação exibindo o conteúdo

Passamos como parâmetro o data que cada sessão deve conter

*const* scrollAnima = new ScrollAnima('[data-anime="scroll"]')

scrollAnima.init()

Com o init, executamos o método para coletar a distancia das sessões, o de checar a distancia assim que abrir o site, para dar display na primeira section, e adicionamos evento de scroll para chrgar distancia

O método de pegar as distancias pega cada section passando por map, e adiciona a um objeto a sessão junto a distancia

O método de checar irá adicionar a classe caso o window Y for maior que a altura da section

export default *class* ScrollAnima {

*constructor*(*sections*) {

    this.sections = document.querySelectorAll(sections); *// seleciona as sections*

    this.windowMetade = window.innerHeight \* 0.6; *// equivalente a 40% da tela*

    this.checkDistance = this.checkDistance.bind(this);

  }

*// função que coleta a posição de cada sessão*

*// utilizando desestruturação para transformar sections de node para array, e passando por cada com map*

*// const offset pegando a distancia do topo de cada section*

*// assim, this.distance retorna um objeto contendo a sessão e a distancia de cada*

  getDistance() {

    this.distance = [...this.sections].map((*section*) => {

*const* offset = section.offsetTop;

      return { *// this.distance retorna um objeto com:*

        element: section, *// cada sessão*

        offset: Math.floor(offset - this.windowMetade), *// e a distancia do topo - 40%*

      };

    });

  }

*//metodo que ira adicionar e remover a classe desejada com base no scroll*

  checkDistance() {

    this.distance.forEach((*item*) => {

      if (window.pageYOffset > item.offset) { *//caso a distancia do scroll for maior que o offset*

        item.element.classList.add("ativo");

      } else if (item.element.classList.contains("ativo")) {

        item.element.classList.remove("ativo");

      }

    });

  }

  init() {

    if (this.sections.length) {

      this.getDistance();*// forma o objeto com informaçoes para o check distance*

      this.checkDistance(); *// verifica a distancia de inicio, para aprecer a primeira section*

      window.addEventListener("scroll", this.checkDistance); *// adiciona o evento scroll e verifica a distancia*

    }

    return this

  }

  stop() { *// metodo para parar o scroll casod esejado*

    window.removeEventListener("scroll", this.checkDistance);

  }

}