```
const bucket = ['coffee', 'chips','vegetables','salt','rice'];

const friedRicePromise = new Promise((resolve,reject)=>{
    if(bucket.includes("vegetables")&& bucket.includes("salt") &&
bucket.includes("rice")){
        resolve({value:"friedrice"});
    }else{
        reject("could not do it");}
})

friedRicePromise
.then(
    (myfriedRice)=>{console.log("lets eat ", myfriedRice);})
.catch(
    (error)=>{console.log(error)})
```

Function returning Promise

```
function ricePromise(){
  const bucket = ['coffee', 'chips','vegetables','salts','rice'];

  return new Promise((resolve,reject)=>{
    if(bucket.includes("vegetables")&& bucket.includes("salt") &&
bucket.includes("rice")){
        resolve({value:"friedrice"});
    }else{
        reject("could not do it");
    }})}
```

- Use Promise

```
ricePromise() //promise name nhi function calling
.then(
  (myfriedRice)=>{console.log("lets eat ", myfriedRice);})
.catch(
  (error)=>{console.log(error)})
```

- Resolve/ Reject Promise after 2 seconds

```
function myPromise(){
    return new Promise((resolve, reject)=>{
        const value = true;
```

```
        setTimeout(()=>{
            if(value){resolve();
            }else{reject();}
        },2000)
    })}

myPromise()
    .then(()=>{console.log("resolved")})
    .catch(()=>{console.log("rejected")})
```

Promise.resolve

- It `takes value`
- `Returns a promise that resolves with that value`

```
const myPromise = Promise.resolve(5); //Returns promise that resolves with value 5
myPromise.then(value=>{ // value = 5
  console.log(value); /// 5
})
```

- `return value` ~ `return Promise.resolve(value)` -> value is a promise

then()

- `Always returns a promise`
- Therefore, we can create Promise Chaining
- If we den't return -> `return undefined;` happens internally

Promise Chaining

```
// create promise function
function myPromise(){
  return new Promise((resolve, reject)=>{
    resolve("foo");
  })
}

myPromise()
  .then((value)=>{ // value is the value given when promise resolved
    console.log(value); /// foo
    value += "bar";
    return value // value is a promise*** ~ return Promise.resolve(value)
  })
```

```
  .then((value) =>{
    console.log(value); /// foobar
    value += " store";
    return value; // value is a promise***
  })
  .then(value=>{
    console.log(value); /// foobar store
  })
```

Callback hell to Flat Code-> Using Promise

- Earlier

```
function changeText(element, text, color, time, onSuccessCallback,
onFailureCallback) {
    setTimeout(()=>{
        if(element){
            element.textContent = text;
            element.style.color = color;
            if(onSuccessCallback){onSuccessCallback();}}
        else{
            if(onFailureCallback){onFailureCallback();}}}
    ,time)}

changeText(heading1, "one","violet",1000,()=>{
  changeText(heading2, "two","purple",2000,()=>{
    changeText(heading3, "three","red",1000,()=>{
      changeText(heading4, "four","pink",1000,()=>{
        changeText(heading5, "five","green",2000,()=>{
          changeText(heading6, "six","blue",1000,()=>{
            changeText(heading7, "seven","brown",1000,()=>{
              changeText(heading8, "eight","cyan",1000,()=>{
                changeText(heading9, "nine","#cda562",1000,()=>{
                  changeText(heading10, "ten","dca652",1000,()=>{

                  },()=>{console.log("Heading10 does not exist")})
                },()=>{console.log("Heading9 does not exist")})
              },()=>{console.log("Heading8 does not exist")})
            },()=>{console.log("Heading7 does not exist")})
          },()=>{console.log("Heading6 does not exist")})
        },()=>{console.log("Heading5 does not exist")})
      },()=>{console.log("Heading4 does not exist")})
    },()=>{console.log("Heading3 does not exist")})
  },()=>{console.log("Heading2 does not exist")})
},()=>{console.log("Heading1 does not exist")})
```

- Now

```
const heading1 = document.querySelector(".heading1");
.
.
.
const heading10 = document.querySelector(".heading10");

function changeText(element, text, color, time) {
    return new Promise((resolve, reject) => {
        setTimeout(()=>{
            if(element){
              element.textContent = text;
              element.style.color = color;
              resolve();
            }else{
              reject("element not found");}
          },time)
    })}
changeText(heading1, "one", "red", 1000)
  .then(()=>{
    return changeText(heading2, "two", "purple", 1000)}) // returned for next then
    .
    .
    .
  .then(()=>{
     return changeText(heading10, "ten", "dca652", 1000)})
  .catch((error)=>{
      alert(error);
  })

// or

changeText(heading1, "one", "red", 1000)
  .then(()=>changeText(heading2, "two", "purple", 1000))
  .then(()=>changeText(heading2, "two", "purple", 1000))
  .then(()=>changeText(heading2, "two", "purple", 1000))
  .then(()=>changeText(heading2, "two", "purple", 1000))
  .then(()=>changeText(heading2, "two", "purple", 1000))
  .then(()=>changeText(heading2, "two", "purple", 1000))
  .then(()=>changeText(heading2, "two", "purple", 1000))
  .then(()=>changeText(heading2, "two", "purple", 1000))
  .then(()=>changeText(heading2, "two", "purple", 1000))
  .then(()=>changeText(heading2, "two", "purple", 1000))
  .then(()=>changeText(heading2, "two", "purple", 1000))
  .then(()=>changeText(heading3, "three", "green", 1000))
  .then(()=>changeText(heading10, "ten", "orange", 1000))
  .catch((error)=>{
      alert(error);
  })
```
