**Obrada izuzetaka**

**00:12:41**

Jedinica: 24 od 30

**Rezime**

* Tokom izvršavanja programskog koda bilo kojeg jezika, može doći do pojave greške.
* Izuzetak predstavlja reprezentaciju greške koja se dogodila u kodu.
* Izuzetak se u JavaScriptu predstavlja Error objektom.
* Obrada izuzetaka podrazumeva adekvatno rukovanje Error objektom i sprečavanje prekida izvršavanja koda korišćenjem specijalne try...catch naredbe.
* Pored try i catch blokova, naredba try…catch može da sadrži još jedan dodatni blok, finally, koji se koristi kako bi se definisao kod koji će se obavezno izvršiti.
* Samostalno izbacivanje izuzetka se može postići korišćenjem naredbe throw.

U prethodnim lekcijama ovog modula predstavljeni su različiti načini za kontrolu toka izvršavanja koda. U ovoj lekciji biće opisan još jedan scenario, koji može da izazove uslovno izvršavanje koda. Reč je o obradi izuzetaka.

**Softverske greške**

Tokom izvršavanja programskog koda bilo kojeg jezika, može doći do pojave greške. Greške su sastavni deo svakog programa, i pored velikih napora koji se ulažu u njihovo detektovanje i uklanjanje, veoma je teško proizvesti ozbiljniju aplikaciju, bez koda koji bi u nekom trenutku mogao da izazove grešku.

Greške prilikom izvršavanja koda mogu biti izazvane od strane brojnih faktora. Neke od njih se mogu pripisati manjkavo napisanom kodu, dok druge svoj izvor imaju u spoljnom okruženju. Spoljni faktori koji mogu da izazovu pojavu grešaka prilikom izvršavanja koda mogu da budu nedostatak interne ili eksterne memorije, prekid mrežne konekcije, nekorektan rad nekog od uređaja itd. S obzirom da se JavaScript jezik izvršava unutar browsera ili nekog drugog softverskog proizvoda koji poseduje JS [parser](https://www.link-elearning.com/linkdl/opisPojma.php?id=144352), o ovakvim stvarima brine samo okruženje.

Za programera, mnogo su bitniji interni faktori koji mogu da dovedu do pojave greške prilikom izvršavanja JS koda. Na primer, sledeći primer ilustruje JS kod koji sadrži grešku:

|  |  |
| --- | --- |
| 1 | alert("Value of variable b is : " + b ); |

Funkcija prikazane linije je ispis vrednosti promenljive *b*. Ipak, ukoliko promenljiva*b* nije prethodno deklarisana, navedena linija proizvešće grešku:

|  |  |
| --- | --- |
| 1 | ReferenceError: b is not defined |

Ovo je samo jedan od primera koda koji u JavaScript jeziku može da proizvede grešku. Kada u kodu dođe do ovakve situacije, izvršavanje JavaScript koda se prekida, tako da će sav kod koji se nalazi ispod onog koji je izazvao grešku ostati neizvršen. Ipak, prekidanje izvršavanja koda nije jedino što se događa kada dođe do greške prilikom izvršavanja koda, jer se u takvoj situaciji kreira i *izbacuje*specijalan tip objekta koji se naziva izuzetak.

**Izuzetak (Exception)**

Izuzetak predstavlja reprezentaciju greške koja se dogodila u kodu. Gotovo svi jezici poznaju pojam izuzetka, kojim se olakšava rukovanje greškama u kodu, pa tako ni JavaScript nije izuzetak. Izuzetak se u JavaScriptu predstavlja Error objektom, koji može imati različita svojstva u zavisnosti od implementacije parsera. Ipak, standardna svojstva koja Error objekat uglavnom poseduje su:

* Name
* message

JavaScript poznaje razne specijalizovane vrste Error objekta koji ovaj objekat nasleđuju i tako obezbeđuju detaljnije informacije o različitim tipovima grešaka. Sada je pravi trenutak da se postavi jedno pitanje: koja je svrha postojanja izuzetaka?

Izuzeci su integralni sastojak sistema za rukovanje greškama, i može se reći da je svrha izuzetaka da pruže informaciju o grešci koja se dogodila u kodu, i da pored toga omoguće programu da nastavi izvršavanje. Da bi se tako nešto postiglo, neophodno je pribeći tehnici koja se naziva obrada izuzetaka.

**Obrada izuzetaka**

Obrada izuzetaka podrazumeva adekvatno rukovanje Error objektom i sprečavanje prekida izvršavanja koda. Obrada izuzetaka se postiže korišćenjem specijalne try...catch naredbe. Ova naredba se sastoji iz dva bloka koda, a omogućava da se eventualni izuzetak koji može nastati tokom izvršavanja koda obradi u specijalnom bloku. Sintaksa naredbe try...catch je sledeća:

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | try {      //block of code to try  }  catch(err) {      //block of code to handle errors  } |

Kao što sama imena blokova sugerišu, try omogućava da se definiše blok koda koji će pokušati da se izvrši. Ukoliko tokom izvršavanja takvog koda dođe do greške, izvršavanje se prebacuje na catchblok.

Uvidom u sintaksu try…catch bloka, može se videti da catch blok prihvata jedan parametar koji je obeležen sa err. Ovo je upravo promenljiva koja sadrži objekat tipa Error. Naziv err je proizvoljan i može se menjati.

Try…catch blok, u praktičnoj upotrebi:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | try {   alert("Value of variable b is : " + b );  }  catch(err) {   console.log('Error name: ' + err.name);   console.log('Error message: ' + err.message);  } |

Kod koji je nešto ranije napisan kao demonstracija koda koji proizvodi grešku sada je smešten unutar try bloka. Na taj način će biti *pokušano* njegovo izvršavanje. S obzirom da kod proizvodi grešku, takva greška se hvata korišćenjem catch bloka, čija promenljiva err dobija vrednost objekta Error. Unutar catch bloka se vrši čitanje njegova dva svojstva name i messagei ispis njihovih vrednosti u konzoli. Tako prikazani kod proizvodi sledeći izlaz:

|  |  |
| --- | --- |
| 1  2 | Error name: ReferenceErrortest  Error message: b is not defined |

Upravo prikazani primer predstavlja osnovni način za rukovanje greškama koje se mogu dogoditi prilikom izvršavanja koda. Umesto da se izvršavanje koda prekine, uz pomoć try…catch naredbe dobija se informacija o grešci, a izvršavanje se nastavlja na prvoj sledećoj liniji.

Pored try i catch blokova, naredba try…catch može da sadrži još jedan dodatni blok – finally. Blok finally se koristi kako bi se definisao kod koji će se obavezno izvršiti, odmah nakon obrade izuzetka, bez obzira da li je do greške došlo. Sintaksa je sledeća:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | try {      Block of code to try  }  catch(err) {      Block of code to handle errors  }  finally{  Block of code to always execute  } |

Kod definisan finally blokom uvek se izvršava, bez obzira da li je do greške došlo ili ne. Česta upotreba ovog bloka jeste oslobađanje resursa koji mogu biti rezervisani. Jednostavno, ukoliko je došlo do greške, takvi resursi više nisu potrebni, pa ih je najbolje osloboditi.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | try {   alert("Value of variable b is : " + b );  }  catch(err) {   console.log('Error name: ' + err.name);   console.log('Error message: ' + err.message);  }  finally{  console.log('This message will always be shown');  } |

Kod proizvodi izlaz:

|  |  |
| --- | --- |
| 1  2  3 | Error name: ReferenceErrortest  Error message: b is not defined  This message will always be shown |

**Unutar kog bloka se postavlja kod koji može da proizvede izuzetak?**

 Try

 Catch

 Finally

 ništa od navedenog

**Izbacivanje izuzetaka**

U do sada prikazanim primerima, posao izbacivanja izuzetaka obavljalo je samo okruženje. Ipak, moguće je i samostalno izvršiti izbacivanje izuzetka i to korišćenjem naredbe throw. Sintaksa ove naredbe je:

|  |  |
| --- | --- |
| 1 | throw expression; |

Naredba throw formira se korišćenjem ključne reči throw, kojoj sledi izraz sa vrednošću koju je potrebno izbaciti. Naredbom throw može se izbaciti vrednost bilo kojeg tipa:

|  |  |
| --- | --- |
| 1  2  3 | throw "This is Error 666";   // String type  throw 36;         // Number type  throw false;       // Boolean type |

Ipak, najčešća je praksa izbacivanje izuzetaka objektnog tipa, koji su specijalno dizajnirani za tu namenu. Najopštiji tip izuzetka u JavaScriptu je objektni tip Error. Pored ovog tipa izuzetka, JavaScript poznaje još nekoliko ugrađenih objekata za predstavljanje izuzetaka. Svi oni su prikazani u tabeli 24.1.

|  |  |
| --- | --- |
| **Tip greške** | **Opis** |
| Error | osnovni objekat za predstavljanje grešaka |
| RangeError | objekat koji predstavlja grešku izazvanu prekoračenjem opsega; javlja se u situacijama kada se prekorači opseg niza, ili kada se metodama koje prihvataju numeričke vrednosti prosledi neodgovarajuća vrednost |
| ReferenceError | greška izazvana referenciranjem nepostojeće promenljive |
| SyntaxError | objekat koji predstavlja grešku izazvanu pokušajem interpretracije sintaksno pogrešno napisanog koda |
| TypeError | objekat koji predstavlja grešku do koje dolazi kada je vrednost koja učestvuje u nekom izrazu neočekivanog tipa |
| URIError | objekat koji predstavlja grešku do koje dolazi kada se globalnoj funkciji URI prosledi nekorektan URI |

*Tabela 24.1 – Tipovi izuzetaka u JS-u*

Primer izbacivanja izuzetka korišćenjem naredbe throw biće ilustrovan na primeru deljenja broja nulom. Deljenje nekog broja nulom u matematici nije dozvoljeno, a neki jezici tako nešto sankcionišu generisanjem odgovarajućeg izuzetka. JavaScript nije jedan od takvih jezika, zbog postojanja specijalne numeričke vrednosti Infinity. Ipak, u sledećem primeru biće prikazan kod koji će u ovakvim situacijama generisati izuzetak:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16 | var x = 5;  var y = 0;  var z;    try {   if(y == 0)   {    throw Error('Division by zero is not allowed');   }else{    z = x / y;   }   }  catch(err) {   console.log('Error name: ' + err.name);   console.log('Error message: ' + err.message);  } |

Na početku primera deklarisane su tri promenljive: x, y i z. Promenljive x i y su inicijalizovane vrednostima 5 i 0 respektivno. Unutar try bloka, smeštena je logika za obavljanje deljenja vrednosti promenljive x promenljivom y. Ali, pored linije koja obavlja pomenuto deljenje, definisan je i jedan uslovni blok, čija je uloga da proveri vrednosti delioca. Ukoliko je delilac jednak nuli, podiže se izuzetak naredbom throw. Izuzetak se predstavlja objektom Error, a konstruktoru objekta se prosleđuje poruka greške. Tako će prikazani kod proizvesti sledeći rezultat:

|  |  |
| --- | --- |
| 1  2 | Error name: Error  Error message: Division by zero is not allowed |

Jasno je da je parametar koji je prosleđen konstruktoru objekta Error iskorišćen za postavljanje vrednosti promenljive message. Prilikom izbacivanja izuzetka, moguće je postaviti vrednosti i promenljive name:

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20 | var x = 5;  var y = 0;  var z;    try {     if(y == 0)   {    var e = new Error('Division by zero is not allowed');    e.name = 'Invalid division';    throw e;     }else{    z = x / y;   }  }  catch(err) {   console.log('Error name: ' + err.name);   console.log('Error message: ' + err.message); }Obrada izuzetaka **00:12:41**  Jedinica: 24 od 30  **Rezime**   * Tokom izvršavanja programskog koda bilo kojeg jezika, može doći do pojave greške. * Izuzetak predstavlja reprezentaciju greške koja se dogodila u kodu. * Izuzetak se u JavaScriptu predstavlja Error objektom. * Obrada izuzetaka podrazumeva adekvatno rukovanje Error objektom i sprečavanje prekida izvršavanja koda korišćenjem specijalne try...catch naredbe. * Pored try i catch blokova, naredba try…catch može da sadrži još jedan dodatni blok, finally, koji se koristi kako bi se definisao kod koji će se obavezno izvršiti. * Samostalno izbacivanje izuzetka se može postići korišćenjem naredbe throw.   U prethodnim lekcijama ovog modula predstavljeni su različiti načini za kontrolu toka izvršavanja koda. U ovoj lekciji biće opisan još jedan scenario, koji može da izazove uslovno izvršavanje koda. Reč je o obradi izuzetaka.  **Softverske greške**  Tokom izvršavanja programskog koda bilo kojeg jezika, može doći do pojave greške. Greške su sastavni deo svakog programa, i pored velikih napora koji se ulažu u njihovo detektovanje i uklanjanje, veoma je teško proizvesti ozbiljniju aplikaciju, bez koda koji bi u nekom trenutku mogao da izazove grešku.  Greške prilikom izvršavanja koda mogu biti izazvane od strane brojnih faktora. Neke od njih se mogu pripisati manjkavo napisanom kodu, dok druge svoj izvor imaju u spoljnom okruženju. Spoljni faktori koji mogu da izazovu pojavu grešaka prilikom izvršavanja koda mogu da budu nedostatak interne ili eksterne memorije, prekid mrežne konekcije, nekorektan rad nekog od uređaja itd. S obzirom da se JavaScript jezik izvršava unutar browsera ili nekog drugog softverskog proizvoda koji poseduje JS [parser](https://www.link-elearning.com/linkdl/opisPojma.php?id=144352), o ovakvim stvarima brine samo okruženje.  Za programera, mnogo su bitniji interni faktori koji mogu da dovedu do pojave greške prilikom izvršavanja JS koda. Na primer, sledeći primer ilustruje JS kod koji sadrži grešku:   |  |  | | --- | --- | | 1 | alert("Value of variable b is : " + b ); |   Funkcija prikazane linije je ispis vrednosti promenljive *b*. Ipak, ukoliko promenljiva*b* nije prethodno deklarisana, navedena linija proizvešće grešku:   |  |  | | --- | --- | | 1 | ReferenceError: b is not defined |   Ovo je samo jedan od primera koda koji u JavaScript jeziku može da proizvede grešku. Kada u kodu dođe do ovakve situacije, izvršavanje JavaScript koda se prekida, tako da će sav kod koji se nalazi ispod onog koji je izazvao grešku ostati neizvršen. Ipak, prekidanje izvršavanja koda nije jedino što se događa kada dođe do greške prilikom izvršavanja koda, jer se u takvoj situaciji kreira i *izbacuje*specijalan tip objekta koji se naziva izuzetak.  **Izuzetak (Exception)**  Izuzetak predstavlja reprezentaciju greške koja se dogodila u kodu. Gotovo svi jezici poznaju pojam izuzetka, kojim se olakšava rukovanje greškama u kodu, pa tako ni JavaScript nije izuzetak. Izuzetak se u JavaScriptu predstavlja Error objektom, koji može imati različita svojstva u zavisnosti od implementacije parsera. Ipak, standardna svojstva koja Error objekat uglavnom poseduje su:   * Name * message   JavaScript poznaje razne specijalizovane vrste Error objekta koji ovaj objekat nasleđuju i tako obezbeđuju detaljnije informacije o različitim tipovima grešaka. Sada je pravi trenutak da se postavi jedno pitanje: koja je svrha postojanja izuzetaka?  Izuzeci su integralni sastojak sistema za rukovanje greškama, i može se reći da je svrha izuzetaka da pruže informaciju o grešci koja se dogodila u kodu, i da pored toga omoguće programu da nastavi izvršavanje. Da bi se tako nešto postiglo, neophodno je pribeći tehnici koja se naziva obrada izuzetaka.  **Obrada izuzetaka**  Obrada izuzetaka podrazumeva adekvatno rukovanje Error objektom i sprečavanje prekida izvršavanja koda. Obrada izuzetaka se postiže korišćenjem specijalne try...catch naredbe. Ova naredba se sastoji iz dva bloka koda, a omogućava da se eventualni izuzetak koji može nastati tokom izvršavanja koda obradi u specijalnom bloku. Sintaksa naredbe try...catch je sledeća:   |  |  | | --- | --- | | 1  2  3  4  5  6 | try {      //block of code to try  }  catch(err) {      //block of code to handle errors  } |   Kao što sama imena blokova sugerišu, try omogućava da se definiše blok koda koji će pokušati da se izvrši. Ukoliko tokom izvršavanja takvog koda dođe do greške, izvršavanje se prebacuje na catchblok.  Uvidom u sintaksu try…catch bloka, može se videti da catch blok prihvata jedan parametar koji je obeležen sa err. Ovo je upravo promenljiva koja sadrži objekat tipa Error. Naziv err je proizvoljan i može se menjati.  Try…catch blok, u praktičnoj upotrebi:   |  |  | | --- | --- | | 1  2  3  4  5  6  7 | try {   alert("Value of variable b is : " + b );  }  catch(err) {   console.log('Error name: ' + err.name);   console.log('Error message: ' + err.message);  } |   Kod koji je nešto ranije napisan kao demonstracija koda koji proizvodi grešku sada je smešten unutar try bloka. Na taj način će biti *pokušano* njegovo izvršavanje. S obzirom da kod proizvodi grešku, takva greška se hvata korišćenjem catch bloka, čija promenljiva err dobija vrednost objekta Error. Unutar catch bloka se vrši čitanje njegova dva svojstva name i messagei ispis njihovih vrednosti u konzoli. Tako prikazani kod proizvodi sledeći izlaz:   |  |  | | --- | --- | | 1  2 | Error name: ReferenceErrortest  Error message: b is not defined |   Upravo prikazani primer predstavlja osnovni način za rukovanje greškama koje se mogu dogoditi prilikom izvršavanja koda. Umesto da se izvršavanje koda prekine, uz pomoć try…catch naredbe dobija se informacija o grešci, a izvršavanje se nastavlja na prvoj sledećoj liniji.  Pored try i catch blokova, naredba try…catch može da sadrži još jedan dodatni blok – finally. Blok finally se koristi kako bi se definisao kod koji će se obavezno izvršiti, odmah nakon obrade izuzetka, bez obzira da li je do greške došlo. Sintaksa je sledeća:   |  |  | | --- | --- | | 1  2  3  4  5  6  7  8  9 | try {      Block of code to try  }  catch(err) {      Block of code to handle errors  }  finally{  Block of code to always execute  } |   Kod definisan finally blokom uvek se izvršava, bez obzira da li je do greške došlo ili ne. Česta upotreba ovog bloka jeste oslobađanje resursa koji mogu biti rezervisani. Jednostavno, ukoliko je došlo do greške, takvi resursi više nisu potrebni, pa ih je najbolje osloboditi.   |  |  | | --- | --- | | 1  2  3  4  5  6  7  8  9  10 | try {   alert("Value of variable b is : " + b );  }  catch(err) {   console.log('Error name: ' + err.name);   console.log('Error message: ' + err.message);  }  finally{  console.log('This message will always be shown');  } |   Kod proizvodi izlaz:   |  |  | | --- | --- | | 1  2  3 | Error name: ReferenceErrortest  Error message: b is not defined  This message will always be shown |     **Unutar kog bloka se postavlja kod koji može da proizvede izuzetak?**   Try   Catch   Finally   ništa od navedenog    **Izbacivanje izuzetaka**  U do sada prikazanim primerima, posao izbacivanja izuzetaka obavljalo je samo okruženje. Ipak, moguće je i samostalno izvršiti izbacivanje izuzetka i to korišćenjem naredbe throw. Sintaksa ove naredbe je:   |  |  | | --- | --- | | 1 | throw expression; |   Naredba throw formira se korišćenjem ključne reči throw, kojoj sledi izraz sa vrednošću koju je potrebno izbaciti. Naredbom throw može se izbaciti vrednost bilo kojeg tipa:   |  |  | | --- | --- | | 1  2  3 | throw "This is Error 666";   // String type  throw 36;         // Number type  throw false;       // Boolean type |   Ipak, najčešća je praksa izbacivanje izuzetaka objektnog tipa, koji su specijalno dizajnirani za tu namenu. Najopštiji tip izuzetka u JavaScriptu je objektni tip Error. Pored ovog tipa izuzetka, JavaScript poznaje još nekoliko ugrađenih objekata za predstavljanje izuzetaka. Svi oni su prikazani u tabeli 24.1.   |  |  | | --- | --- | | **Tip greške** | **Opis** | | Error | osnovni objekat za predstavljanje grešaka | | RangeError | objekat koji predstavlja grešku izazvanu prekoračenjem opsega; javlja se u situacijama kada se prekorači opseg niza, ili kada se metodama koje prihvataju numeričke vrednosti prosledi neodgovarajuća vrednost | | ReferenceError | greška izazvana referenciranjem nepostojeće promenljive | | SyntaxError | objekat koji predstavlja grešku izazvanu pokušajem interpretracije sintaksno pogrešno napisanog koda | | TypeError | objekat koji predstavlja grešku do koje dolazi kada je vrednost koja učestvuje u nekom izrazu neočekivanog tipa | | URIError | objekat koji predstavlja grešku do koje dolazi kada se globalnoj funkciji URI prosledi nekorektan URI |   *Tabela 24.1 – Tipovi izuzetaka u JS-u*  Primer izbacivanja izuzetka korišćenjem naredbe throw biće ilustrovan na primeru deljenja broja nulom. Deljenje nekog broja nulom u matematici nije dozvoljeno, a neki jezici tako nešto sankcionišu generisanjem odgovarajućeg izuzetka. JavaScript nije jedan od takvih jezika, zbog postojanja specijalne numeričke vrednosti Infinity. Ipak, u sledećem primeru biće prikazan kod koji će u ovakvim situacijama generisati izuzetak:   |  |  | | --- | --- | | 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16 | var x = 5;  var y = 0;  var z;    try {   if(y == 0)   {    throw Error('Division by zero is not allowed');   }else{    z = x / y;   }   }  catch(err) {   console.log('Error name: ' + err.name);   console.log('Error message: ' + err.message);  } |   Na početku primera deklarisane su tri promenljive: x, y i z. Promenljive x i y su inicijalizovane vrednostima 5 i 0 respektivno. Unutar try bloka, smeštena je logika za obavljanje deljenja vrednosti promenljive x promenljivom y. Ali, pored linije koja obavlja pomenuto deljenje, definisan je i jedan uslovni blok, čija je uloga da proveri vrednosti delioca. Ukoliko je delilac jednak nuli, podiže se izuzetak naredbom throw. Izuzetak se predstavlja objektom Error, a konstruktoru objekta se prosleđuje poruka greške. Tako će prikazani kod proizvesti sledeći rezultat:   |  |  | | --- | --- | | 1  2 | Error name: Error  Error message: Division by zero is not allowed |   Jasno je da je parametar koji je prosleđen konstruktoru objekta Error iskorišćen za postavljanje vrednosti promenljive message. Prilikom izbacivanja izuzetka, moguće je postaviti vrednosti i promenljive name:   |  |  | | --- | --- | | 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20 | var x = 5;  var y = 0;  var z;    try {     if(y == 0)   {    var e = new Error('Division by zero is not allowed');    e.name = 'Invalid division';    throw e;     }else{    z = x / y;   }  }  catch(err) {   console.log('Error name: ' + err.name);   console.log('Error message: ' + err.message);  } |   Ovoga puta Error objekat je kreiran korišćenjem ključne reči new, a zatim je postavljena vrednost njegovog svojstva name. Error objekat upakovan je u promenljivu e, i emitovan korišćenjem naredbe throw.  Prilikom izbacivanja izuzetaka korišćenjem naredbe throw, moguće je koristiti i bilo koji objekat prikazan u tabeli 24.1.   |  |  | | --- | --- | | 1  2  3  4  5  6  7  8  9  10  11  12 | var x = 5;    try {   if(!(x > 5 && x < 10))   {    throw RangeError("Value must be greater than 5 and less than 10");   }  }  catch(err) {   console.log('Error name: ' + err.name);   console.log('Error message: ' + err.message);  } |   U prikazanom primeru se proverava da li je vrednost promenljive x unutar definisanog opsega. Ukoliko nije, emituje se izuzetak tipa RangeError. Prikazani primer proizvodi efekat:   |  |  | | --- | --- | | 1  2 | Error name: RangeError  Error message: Value must be greater than 5 and less than 10 |   Kao izuzetak se može emitovati i bilo koji korisnički kreiran objekat:   |  |  | | --- | --- | | 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16 | var x = 5;    try {     if(!(x > 5 && x < 10))   {    throw {     err\_name: "Out of range",     reason: "Value must be greater than 5 and less than 10"    }   }  }  catch(err) {   console.log('Error name: ' + err.err\_name);   console.log('Error message: ' + err.reason);  } |   Kod proizvodi sledeći rezultat:   |  |  | | --- | --- | | 1  2 | Error name: Out of range  Error message: Value must be greater than 5 and less than 10 | |

Ovoga puta Error objekat je kreiran korišćenjem ključne reči new, a zatim je postavljena vrednost njegovog svojstva name. Error objekat upakovan je u promenljivu e, i emitovan korišćenjem naredbe throw.

Prilikom izbacivanja izuzetaka korišćenjem naredbe throw, moguće je koristiti i bilo koji objekat prikazan u tabeli 24.1.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12 | var x = 5;    try {   if(!(x > 5 && x < 10))   {    throw RangeError("Value must be greater than 5 and less than 10");   }  }  catch(err) {   console.log('Error name: ' + err.name);   console.log('Error message: ' + err.message);  } |

U prikazanom primeru se proverava da li je vrednost promenljive x unutar definisanog opsega. Ukoliko nije, emituje se izuzetak tipa RangeError. Prikazani primer proizvodi efekat:

|  |  |
| --- | --- |
| 1  2 | Error name: RangeError  Error message: Value must be greater than 5 and less than 10 |

Kao izuzetak se može emitovati i bilo koji korisnički kreiran objekat:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16 | var x = 5;    try {     if(!(x > 5 && x < 10))   {    throw {     err\_name: "Out of range",     reason: "Value must be greater than 5 and less than 10"    }   }  }  catch(err) {   console.log('Error name: ' + err.err\_name);   console.log('Error message: ' + err.reason);  } |

Kod proizvodi sledeći rezultat:

|  |  |
| --- | --- |
| 1  2 | Error name: Out of range  Error message: Value must be greater than 5 and less than 10 |