JavaScript debugging

Greške su sastavni deo gotovo svakog softverskog proizvoda. Vrlo je teško pronaći složeniji program bez greške koja se u nekim situacijama može ispoljiti. Takođe, tokom razvoja i testiranja softverskih proizvoda vrlo često se detektuje i otkloni veliki broj softverskih grešaka. Neke od njih nikada ne budu detektovane i otklonjene.

Greške mogu prouzrokovati brojne probleme prilikom izvršavanja nekog programa. Simptomi mogu varirati od neočekivanog rezultata izvršavanja, preko korumpiranih podataka, pa sve do potpunog otkaza i nemogućnosti korišćenja kompletnog sistema. Stoga je razumevanje pojma softverskih grešaka i načina za njihovo otklanjanje od presudnog značaja prilikom razvoja softvera.

U jednoj od prethodnih lekcija ovoga kursa već je spomenut pojam grešaka, ali iz ugla njihove objektne reprezentacije – izuzetaka. Prikazano je kako se izuzeci obrađuju i time sprečava prestanak rada skripte. U ovoj lekciji će softverskim greškama u JavaScriptu biti posvećeno više pažnje, sa posebnim akcentom na različite načine za detekciju i otklanjanje tih grešaka.

Softverske greške

Greška prilikom izvršavanja nekog programa drugačije se naziva **bug**. Naziv dolazi od engleske reči, koja u prevodi znači buba. Pojam je popularizovala američka kompjuterska naučnica Grejs Hoper (Grace Hopper), koja je 1947. godine, pokušavajući da otkloni softversku grešku na harvardskom elektromehaničkom kompjuteru, pronašla moljca zaglavljenog u releju.

Softverska greška jeste bilo koji nedostatak ili otkaz u kompjuterskom programu koji proizvodi netačno ili neočekivano ponašanje. Iz ugla modernih programskih jezika, pa samim tim i frontend programiranja, softverske greške se mogu podeliti u dve grupe:

- sintaksne greške i
- logičke greške.

Sintaksne greške nastaju usled nepoštovanja sintaksnih pravila jezika koji se koristi. Ovakvu vrstu grešaka je uglavnom najjednostavnije detektovati i otkloniti, pošto je program sa sintaksnom greškom delimično ili potpuno nefunkcionalan. Ukoliko se sintaksne greške posmatraju iz ugla frontend razvoja, JavaScript skripta prestaje da se izvršava onoga trenutka kada izvršavanje dođe do naredbe sa sintaksnom greškom. Pri tome, pojava sintaksne greške u JavaScriptu propraćena je i izbacivanjem izuzetka sa odgovarajućom porukom. Stoga je za detekciju i uklanjanje sintaksnih grešaka dovoljan neki od modernih web pregledača i razumevanje poruka koje se dobijaju od JavaScript izvršnog okruženja.

Logičke greške mnogo je teže detektovati i otkloniti. Ova vrsta grešaka nastaje u situacijama u kojima skripta ne proizvodi očekivani rezultat, a JavaScript izvršno okruženje ne prijavljuje bilo kakav izuzetak.

Drugim rečima, kôd sa logičkom greškom ne obavlja ono što se od njega očekuje. Zbog nepostojanja bilo kakve poruke od JavaScript izvršnog okruženja, detekcija i uklanjanje logičkih grešaka zahteva pažljivu analizu koda i poznavanje korišćenja nekog od alata za debug. Na sreću, svi moderni web pregledači poseduju takve alate.

Pitanje

Koje greške su teže za detektovanje?

- **Logičke**
- Sintaksne

Objašnjenje:

Logičke greške mnogo je teže detektovati i otkloniti, zato što ova vrsta grešaka nije propraćena izbacivanjem izuzetka.

Referentni primer

U nastavku lekcije za demonstraciju detekcije i otklanjanja sintaksnih grešaka biće korišćen jedan jednostavan primer, unutar koga postoji nekoliko grešaka obe spomenute vrste. Kompletan kôd HTML stranice izgleda ovako:

```
<!DOCTYPE html>
<html lang="en">

	<head>
		<meta charset="UTF-8">
		<meta name="viewport" content="width=device-width, initial-
		scale=1.0">
		<title>JavaScript Debugging</title>
	</head>

	<body>

		<script>

			var number_a = prompt("Please enter first number:");
			var number_b = prompt("Please enter second number:");

		if(number_a > number_b) {
				alert("Number " + number_a " is greater then " +
				number_b);
			} else {
				alert("Number " + number_b + " is greater then "
				+ number_c);
	}

		</script>

	</body>

</html>
```

Upravo prikazani HTML dokument sadrži jednostavnu skriptu, koja od korisnika zahteva unos dva broja. To se postiže korišćenjem ugrađene funkcije `prompt()`, koja kao efekat ima prikaz modalnog prozora sa poljem unutar koga korisnik treba da unese traženu vrednost.

Brojevi koje korisnik unese smeštaju se unutar promenljivih `number_a` i `number_b`. Skripta zatim treba da utvrdi koji od dva uneta broja je veći i da obaveštenje prikaže korisniku u formi modalnog prozora koji se dobija pozivanjem ugrađene funkcije `alert()`.

Detekcija i uklanjanje sintaksnih grešaka

Ukoliko prikazani HTML dokument otvorite unutar nekog od web pregledača, moći ćete da vidite da se ništa ne dešava. Razlog je i više nego jednostavan. JavaScript kôd na stranici poseduje sintaksnu grešku. Kako takvo nešto znamo?

Pregledom HTML dokumenta unutar web pregledača na prvi pogled se ne dobija nikakva informacija o razlogu nefunkcionisanja skripte. Ipak, informacije je potrebno potražiti unutar konzole koju poseduju svi moderni web pregledači.

Konzola web pregledača

Konzole web pregledača omogućavaju uvid u unutrašnje izvršavanje JavaScript koda. Unutar konzole se štampaju poruke koje generiše izvršno okruženje tokom izvršavanja koda. Pored toga, i programer ima mogućnost da unutar konzole ispiše proizvoljne poruke tokom izvršavanja skripte.

Različiti web pregledači poseduju različite načine za dolazak do konzole.

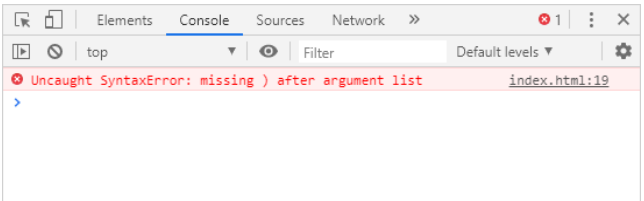
Chrome: iz glavnog menija je potrebno odabrati *More Tools -> Developer Tools*, a zatim iz panela sa razvojnim alatima odabrati tab *Console.*

Firefox: u glavnom meniju je potrebno odabrati *Web Developer -> Web Console.*

Safari: konzola se aktivira iz Develop menija, koji se nalazi u glavnom meniju, uglavnom na vrhu displeja; ukoliko Develop opcije nema unutar glavnog menija kada je Safari pregledač aktivan, potrebno je čekirati *Show Develop menu in menu bar* unutar prozora koji se dobija klikom na opciju *Safari -> Preferences.*

Edge: iz glavnog menija je potrebno odabrati *More Tools -> Developer Tools*, a zatim iz panela sa razvojnim alatima, tab *Console.*

Uvidom u konzolu dobija se informacija o uzroku nefunkcionisanja skripte iz referentnog primera (slika 18.1).



Slika 18.1. Prikaz greške unutar konzole web pregledača

Poruka koja se dobija unutar konzole i više je nego jasna:

```
Uncaught SyntaxError: missing ) after argument list index.html:19
```

Prikazana poruka se može protumačiti na sledeći način: u liniji 19, dokumenta `index.html`, došlo je do pojave izuzetka tipa `SyntaxError` koji nije obrađen. Propratna poruka izuzetka je: *missing) after argument list.*

Kako bi se prikazani problem rešio, potrebno je analizirati liniju 19 (slika 18.2).

```
1  <!DOCTYPE html>
2  <html lang="en">
3
4      <head>
5          <meta charset="UTF-8">
6          <meta name="viewport" content="width=device-width, initial-scale=1.0">
7          <title>JavaScript Debugging</title>
8      </head>
9
10     <body>
11
12         <script>
13
14             var number_a = prompt("Please enter first number:");
15             var number_b = prompt("Please enter second number:");
16
17
18             if(number_a > number_b) {
19                 alert("Number " + number_a " is greater then " + number_b);
20             } else {
21                 alert("Number " + number_b + " is greater then " + number_c);
22             }
23
24         </script>
25
26     </body>
27
28 </html>
```

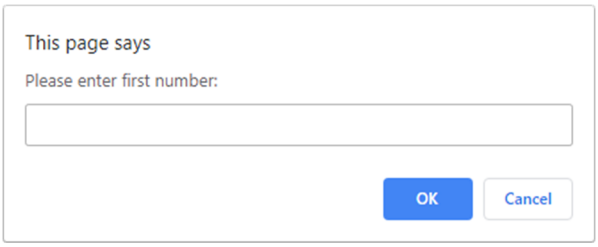
Slika 18.2. Linija 19, unutar koje se nalazi sintaksna greška

Sintaksne greške vrlo često nastaju usled grešaka u kucanju koda, prilikom izostavljanja ili navođenja suvišnih karaktera. Naredbom u liniji 19 treba da se obavi ispis poruke za korisnika. Poruka se formira nadovezivanjem više tekstualnih delova, a između vrednosti promenljive `number_a` i teksta *is greater then*, izostavljen je karakter za nadovezivanje +.

Ovo je klasičan primer sintaksne greške, koja ne dozvoljava da skripta bude parsirana na pravi način. Zbog toga, još prilikom samog učitavanja stranice dolazi do greške, odnosno do emitovanja izuzetka od JavaScript izvršnog okruženja.

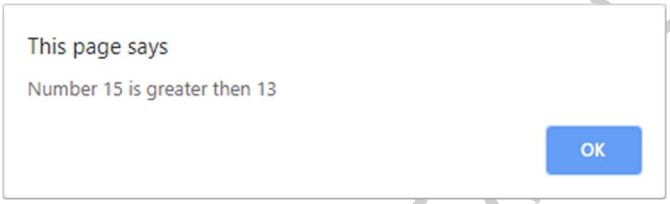
Dodavanjem karaktera za konkatenaciju (+) između promenljive `number_a` i teksta *is greater then,* ispravlja se sintaksna greška koja je sprečavala parsiranje skripte.

Osvežavanjem stranice dolazi do prikaza modalnog prozora za unos vrednosti, što znači da je skripta uspešno parsirana i njeno izvršavanje započelo (slika 18.3).



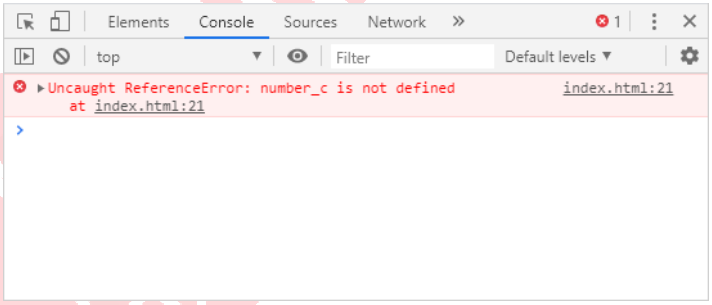
Slika 18.3. Nakon uklanjanja sintaksne greške, skripta se uspešno parsira

Ipak, nisu sve sintaksne greške ovoliko lako uočljive kao upravo ilustrovana greška. U to se možemo uveriti ukoliko testiramo funkcionisanje skripte za poređenje dva broja. Skripta od korisnika zahteva unos dva broja, koja se zatim porede. Kada se na primer unesu brojevi 15 i 13, sve funkcioniše bez greške (slika 18.4).



Slika 18.4. Poruka koja se dobija od skripte nakon uspešno izvršene logike

Ipak, ukoliko funkcionisanje skripte testirate sa nešto drugačijim ulaznim podacima, tako da prvi uneti broj bude manji od drugog (na primer, 13 i 15), unutar web pregledača se ne dobija nikakva poruka. Objašnjenje ćemo opet potražiti unutar konzole (slika 18.5).



Slika 18.5. Poruka o grešci koja se dobija unutar konzole

Poruka sa slike 18.5. koja se dobija unutar konzole može se protumačiti na sledeći način: u 21. liniji dokumenta `index.html` došlo je do neobrađenog izuzetka tipa `ReferenceError`. Propratna poruka izuzetka je: *number_c is not defined*.

Poruka je i više nego jasna. U 21. liniji navedena je promenljiva sa nazivom `number_c`, a da prethodno ona nije deklarisana. Reč je o još jednom klasičnom tipu sintaksnih grešaka, zato što je očigledno napravljena greška prilikom kucanja – umesto `number_a`, napisano je `number_c`. Ispravkom ove štamparske greške skripta iz referentnog primera oslobođena je svih sintaksnih grešaka.

Tipovi sintaksnih grešaka

Iz prethodnih redova mogli ste da vidite da nisu sve sintaksne greške iste. Neke ne dozvoljavaju da se obavi parsiranje JavaScript koda, dok druge kao posledicu imaju emitovanje izuzetaka tek kada izvršavanje dođe do naredbi u kojima se greške nalaze. Ipak, ono što je zajedničko za sve vrste sintaksnih grešaka jeste emitovanje odgovarajućeg izuzetka upakovanog u jedan od sedam (7) sledećih objekata:

- `EvalError` – greška koja nastaje prilikom izvršavanja globalne funkcije eval(),
- `InternalError` – greška koja je nastala interno, unutar JavaScript izvršnog okruženja,
- `RangeError` – greška do koje dolazi kada je numerička vrednost izvan dozvoljenog opsega,
- `ReferenceError` – greška do koje dolazi prilikom neadekvatnog referenciranja, kada se na primer pokušava koristiti promenljiva koja nije deklarisana,
- `SyntaxError` – greška do koje dolazi kada JavaScript parser detektuje sintaksno netačan kôd,
- `TypeError` – greška do koje dolazi kada podatak nije očekivanog tipa,
- `URIError` – greška do koje dolazi kada funkcije encodeURI() i decodeURI() dobiju neodgovarajuće parametre.

Logičke greške

U dosadašnjem toku ove lekcije bilo je reči o sintaksnim greškama u JavaScript kodu. Detekcija i uklanjanje logičkih grešaka zahteva nešto više strpljenja, znanja i iskustva. Logičke greške ne proizvode izuzetke, pa je stoga mnogo teže detektovati neregularno stanje sistema. Ukoliko to do sada niste primetili, i referentni primer iz ove lekcije sadrži nekoliko logičkih grešaka, koje se mogu otkriti temeljnim testiranjem. U nastavku će biti ilustrovani različiti slučajevi korišćenja skripte iz ove lekcije, kako bi se detektovala logička greška.

Slučaj 1

Ulazni podaci: `13, 15`
Poruka: `Number 15 is greater then 13`

Slučaj 2

Ulazni podaci: `15, 13`
Poruka: `Number 15 is greater then 13`

Slučaj 3

Ulazni podaci: `5, 15`
Poruka: `Number 5 is greater then 15`

Slučaj 4

Ulazni podaci: `5, 5`
Poruka: `Number 5 is greater then 5`

Analizom navedenih slučajeva korišćenja može se dosta toga zaključiti o unutrašnjem izvršavanju skripte. U prvom slučaju, uneti su brojevi 13 i 15. Skripta obaveštava da je broj 15 veći od broja 13, na osnovu čega se može zaključiti da program uspešno funkcioniše. I drugi slučaj je sličan, ulazni podaci su identični, ali su uneti obrnutim redom. Skripta emituje identičnu poruku, pa se može zaključiti da i na osnovu *Slučaja 2* program funkcioniše korektno.

Problemi započinju *Slučajem 3*. Uneti brojevi su 5 i 15, a na izlazu se dobija poruka da je broj 5 veći od broja 15. *Slučaj 3* pokazuje da skripta definitivno poseduje određene nedostatke.

Slučaj 4 pokazuje još jedan nedostatak skripte. Uneti brojevi su 5 i 5, a skripta obaveštava da je broj 5 veći od broja 5.

Iako je uzrok logičkih grešaka u prikazanom referentnom primeru moguće naslutiti pažljivijom opservacijom koda, u nastavku lekcije će biti prikazano nekoliko načina za lak pronalazak logičkih grešaka, korišćenjem ugrađenih alata web pregledača.

Korišćenje konzole za logovanje unutrašnjeg stanja skripte

Kako bi se otkrio uzrok logičkih grešaka, neophodno je razumeti unutrašnji tok izvršavanja koda. Za bolje razumevanje stanja u kojima se skripta nalazi tokom izvršavanja, moguće je iskoristiti pomagala web pregledača. Najjednostavnije pomagalo je konzola.

Konzola web pregledača u kombinaciji sa ugrađenim `console` objektom može se koristiti za logovanje informacija o izvršavanju skripte. Na primer, može se napisati nešto ovakvo:

```
var number_a = prompt("Please enter first number:");
var number_b = prompt("Please enter second number:");

console.log("Number A value: " + number_a + ". Number A type: " +
typeof(number_a));
console.log("Number B value: " + number_b + ". Number B type: " + typeof
(number_b));

if(number_a > number_b) {
          alert("Number " + number_a + " is greater then " +
          number_b);
     } else {
          alert("Number " + number_b + " is greater then " +
          number_a);
}
```

Nakon naredbi za preuzimanje vrednosti od korisnika, postavljene su dve naredbe za logovanje vrednosti i tipova promenljivih `number_a` i `number_b`.

Na taj način će unutar konzole moći da se vide neke informacije koje nisu vidljive na samoj stranici:

```
Number A value: 5. Number A type: string
Number B value: 15. Number B type: string
```

Logovanje informacija unutar konzole omogućava nam uvid u osobine promenljivih `number_a` i `number_b`, nakon što korisnik unese vrednosti.

Bitno je primetiti da su, nakon inicijalizacije, promenljive `number_a` i `number_b` tipa `string`.

Poznajući osobine tipova podataka i operatora (o čemu je bilo reči u jednoj od prethodnih lekcija), može se zaključiti da je logička greška u skripti prouzrokovana neodgovarajućim tipovima promenljivih `number_a` i `number_b`. Jednostavno, po trenutnoj logici, obavlja se poređenje dve tekstualne vrednosti, iako je zamišljeno da skripta poredi numeričke vrednosti. Iz lekcije o operatorima poznato je da se poređenje `string` podataka obavlja znatno drugačije od poređenja brojeva. Stoga je za uklanjanje prve logičke greške iz koda potrebno dodati logiku za konverziju `string` vrednosti u `number`:

```
var number_a = prompt("Please enter first number:");
var number_b = prompt("Please enter second number:");

number_a = parseInt(number_a);
number_b = parseInt(number_b);
```

Za konverziju `string` podataka u `number` iskorišćena je `parseInt()` funkcija, kojoj su prosleđene vrednosti koje su preko `prompt()` funkcije dobijene od korisnika. Nakon ove izmene, logička greška iz nešto ranije navedenog *Slučaja 3* više ne postoji:

Slučaj 3

Ulazni podaci: `5, 15`
Poruka: `Number 15 is greater then 5`

Debug alati unutar web pregledača

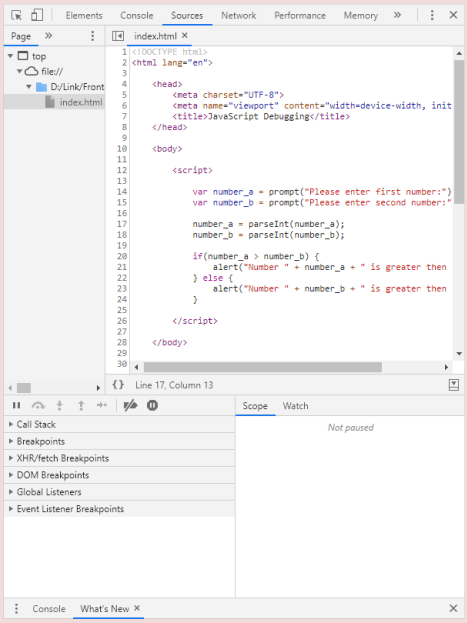
U prethodnim redovima, uz pomoć konzole web pregledača i `console` objekta JavaScript jezika, uspešno je detektovana i otklonjena jedna logička greška iz referentnog primera. Ipak, logička greška iz nešto ranije navedenog *Slučaja 4* još uvek nije korigovana. Jednostavno, kada se unesu dva identična broja, skripta ne funkcioniše kako treba. Kako bismo razumeli izvor problema, ovoga puta će biti iskorišćen nešto drugačiji pristup koji podrazumeva upotrebu najmoćnijeg alata za detekciju grešaka – debuggera web pregledača.

Chrome DevTools Debugger

Svi moderni web pregledači poseduju ugrađene alate za lakšu detekciju grešaka. Oni se nazivaju debuggeri. Po izgledu i osobinama su gotovo identični, a u nastavku će biti ilustrovan takav alat iz Google Chrome web pregledača.

Debugger Chrome pregledača sastavni je deo skupa alata za programere koji se naziva DevTools. To je isti onaj skup alata unutar koga se nalazi i konzola koja je do sada već više puta korišćena. Debuggeru Chrome pregledača pristupa se otvaranjem Sources taba unutar

DevToolsa (slika 18.6).



Slika 18.6. – Debugger Google Chrome pregledača

Otvaranje *Sources* taba ne donosi mnogo toga samo po sebi. Prava moć svakog debuggera ogleda se u mogućnosti pauziranja i kontrolisanja izvršavanja koda. Na taj način je moguće steći uvid u unutrašnju logiku, praćenjem toka izvršavanja koda. Pauziranje izvršavanja koda, te ulazak u specijalni mod za kontrolisanje moguće je postići na dva načina:

- upotrebom ključne reči `debugger` i
- korišćenjem tačaka prekida.

Ključna reč debugger

JavaScript poznaje specijalnu naredbu `debugger`, kojom je moguće aktivirati raspoloživi debugger JavaScript izvršnog okruženja. U našem primeru, upotreba ove ključne reči bi izgledala ovako:

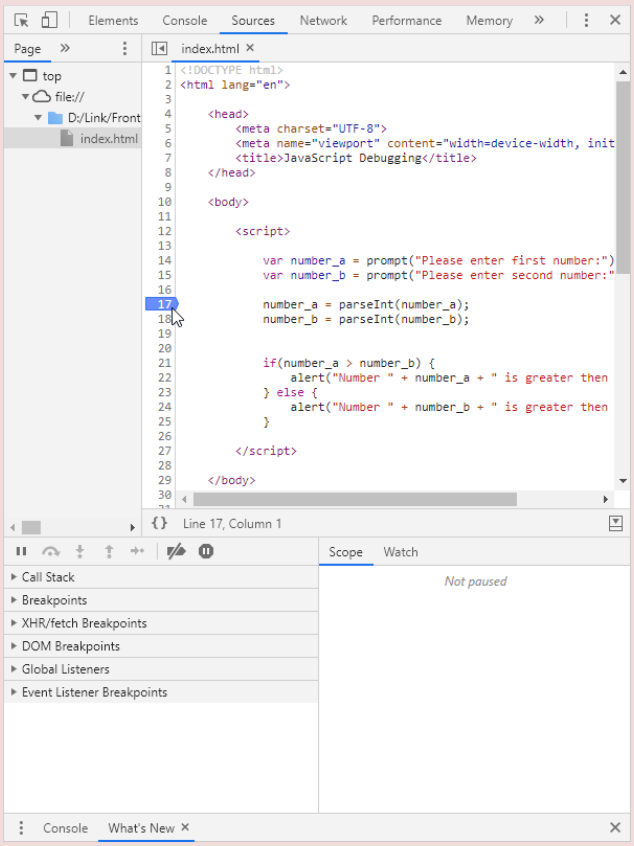
```
var number_a = prompt("Please enter first number:");
var number_b = prompt("Please enter second number:");

debugger;

number_a = parseInt(number_a);
number_b = parseInt(number_b);
```

Naredba `debugger` se može postaviti bilo gde u kodu. U prikazanom primeru ona je postavljena odmah nakon naredbi kojima se od korisnika preuzimaju vrednosti, a pre naredbi za njihovo konvertovanje u podatke tipa `number`. Kako bi naredba debugger imala efekta, unutar web pregledača je potrebno otvoriti debuger.

Onoga trenutka kada izvršavanje koda dođe do ove naredbe, izvršavanje će se pauzirati.

Tačke prekida

Još jedan način za pauziranje izvršavanja JavaScript koda jeste postavljanje tačaka prekida direktno unutar debuggera web pregledača (slika 18.7).



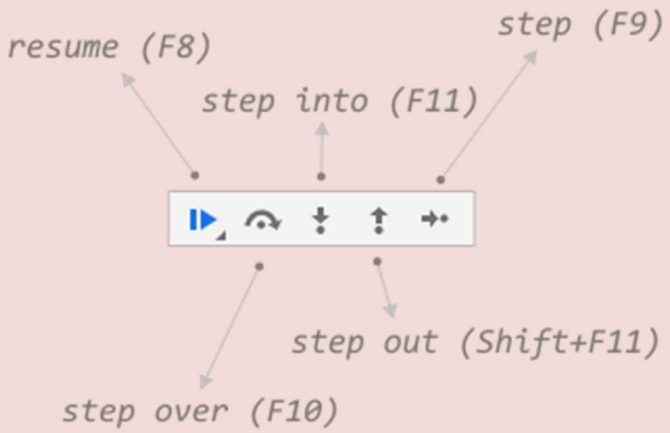
Slika 18.7. – Postavljanje tačke prekida

Slika 18.7. ilustruje postavljanje tačke prekida na liniji 17. Tačka prekida se postavlja jednostavnim klikom na broj linije sa leve strane pregleda izvornog koda stranice. Kao potvrda toga da je tačka prekida uspešno postavljena, broj linije biva označen karakterističnom plavom oznakom. Nakon postavljanja tačke prekida u primeru iz ove lekcije, potrebno je osvežiti stranicu i izvršavanje koda će biti pauzirano u definisanoj tački prekida.

Korišćenjem dva upravo prikazana pristupa postiže se identično – izvršavanje se pauzira u definisanoj tački prekida ili naredbi `debugger`. Nakon što je izvršavanje pauzirano, programer na raspolaganju ima skup kontrola koje može koristiti za rukovođenje izvršavanjem.

Kontrole za rukovanje debuggerom

Za kontrolisanje izvršavanja koda nakon pauziranja unutar debuggera moguće je koristiti nekoliko različitih komandi (slika 18.8).



Slika 18.8. – Opcije za kontrolisanje izvršavanja koda unutar debuggera

Resume – nastavlja normalno izvršavanje koda (bez pauziranja), sve do naredne tačke prekida ili naredbe `debugger`.

Step Over – prelazi na izvršavanje prve sledeće linije, ipak ukoliko je sledeća linija poziv neke funkcije, takva funkcija se izvršava u celosti, a izvršavanje se ponovo pauzira na prvoj naredbi nakon poziva funkcije.

Step Into – prelazi na izvršavanje sledeće linije, ali ukoliko je sledeća linija poziv funkcije, takva funkcija se ne izvršava u celosti, već se izvršavanje pauzira na prvoj naredbi unutar funkcije.

Step Out – ukoliko se izvršavanje nalazi unutar funkcije, izvršava se kompletan ostatak funkcije, a izvršavanje se pauzira na prvoj naredbi nakon poziva takve funkcije.

Step – identično ponašanje kao i kod naredbe step into, s tim što se ne obavlja ulazak u asinhrone funkcije.

Korišćenjem upravo prikazanih komandi može se kontrolisati izvršavanje koda kada se aktivira debugger unutar web pregledača.

Uz pomoć upravo ilustrovanih alata za debugovanje, unutar web pregledača veoma lako se može detektovati još jedna logička greška referentnog primera iz ove lekcije. Ona se odnosi na nešto ranije navedeni *Slučaj 4*, odnosno na scenario u kome korisnik unosi dva identična broja.

U takvoj situaciji pauziranjem izvršavanja i prolaskom kroz kôd, liniju po liniju, može se utvrditi da dolazi do aktiviranja `else` uslovnog bloka, koji emituje poruku koja ne odgovara ulaznim podacima programa.

Kako bi se ovakva logička greška uklonila, neophodno je delimično izmeniti logiku za poređenje:

```
if(number_a > number_b) {
            alert("Number " + number_a + " is greater then " +
            number_b);
      } else if (number_b > number_a) {
            alert("Number " + number_b + " is greater then " +
            number_a);
      } else {
            alert("Numbers are equal.");
            }
```

Sada je dodat još jedan `else if` blok unutar koga se utvrđuje da li je drugi broj veći od prvog. Takođe, izmenjen je i blok `else`, pa se sada on aktivira samo ukoliko su uneti brojevi jednaki.

Rezime

- Softverska greška jeste bilo koji nedostatak ili otkaz u kompjuterskom programu koji proizvodi netačno ili neočekivano ponašanje.
- Greška prilikom izvršavanja nekog programa drugačije se naziva *bug.*
- Sintaksne greške nastaju usled nepoštovanja sintaksnih pravila jezika koji se koristi.
- Logičke greške nastaju u situacijama u kojima skripta ne proizvodi očekivani rezultat, a JavaScript izvršno okruženje ne prijavljuje bilo kakav izuzetak.
- Konzole web pregledača omogućavaju uvid u unutrašnje izvršavanje JavaScript koda.
- Zajedničko za sve vrste sintaksnih grešaka jeste emitovanje odgovarajućeg izuzetka upakovanog u jedan od sedam ugrađenih objekata JavaScript jezika.
- Logičke greške su mnogo teže za detektovanje od sintaksnih.
- Svi moderni web pregledači poseduju ugrađene alate za lakšu detekciju grešaka, oni se nazivaju debuggeri.
- Prava moć svakog debuggera ogleda se u mogućnosti pauziranja i kontrolisanja izvršavanja koda.
- Pauziranje koda unutar JavaScript debuggera može se obaviti na dva načina: korišćenjem naredbe `debugger` i upotrebom tačaka prekida.