Programsko inženjerstvo - Vježbe

4. Strukture podataka

(prilagođeno prema knjizi Eloquent Javascript)

Upoznali smo brojke, *boolean* i *string* -ove kao atomarne tipove podataka. Međutim podaci koji nas okružuju organizirani su i međusobno isprepleteni u složenije strukture. U ovom poglavlju upoznat ćemo dvije osnovne strukture organizacije atomarnih podataka: *object* i *array* .

Polje

Polje, odnosno `array` služi za grupiranje podataka na način da su oni slijedno organizirani. Primjerice, želimo spremiti niz brojki, stringova ili drugih tipova te pritom sačuvati redoslijed pohrane. Uz to dolazi i mogućnost adresiranja određenih elemenata po njihovoj poziciji (prvi, zadnji, 10-ti, *n* -ti, ...).

Primjer:

```
let listaBrojeva = [2, 34, 12, 11, 11, 11, 0];

console.log(listaBrojeva[0]);
console.log(listaBrojeva[2]);
console.log(listaBrojeva[100]);
```

Prvi element indeksiran je sa `0` , a ukoliko adresiramo element koji nije sadržan ne dobivamo grešku već `undefined` vrijednost.

Svojstva

Do sada smo susreli funkciju `console.log` koja je možda izgledala sumnjivo. Zašto?

Naime ona u sebi sadrži točku (`.`) koja u Javascriptu ne može biti sastavni dio naziva, već ona označava da u vrijednosti `console` postoji **svojstvo** (engl. *property*) `log` koje je po svom tipu funkcija, i to funkcija koja služi ispisu u konzolu. Dakle pomoću točke pristupamo svojstvima sadržanim u objektu `console` .

Specifičnost JavaScripta je da gotovo sve vrijednosti mogu imati svoja svojstva za razliku od objektno-orijentiranih jezika gdje je to dozvoljeno samo instancama ili objektima. Jedino `undefined` i `null` ne posjeduju svojstva. Samim time moguće je sljedeće:

```
let a = 5;
console.log(a.toString());
let b = "javascript";
console.log(b.length);
```

Neka važnija ugrađena svojstva:

- `toString()` metoda za pretvorbu u `string`
- `length` svojstvo duljine niza ili stringa

Drugi način na koji se može pristupiti svojstvima jest pomoću uglatih zagrada `[` i `]` :

```
let a = 5;
console.log(a["toString"]());

let b = "javascript";
console.log(b["length"]);
console.log(b["len" + "gth"]);

let svojstvo = "length";
console.log(b[svojstvo])
```

Primjetimo **bitnu** razliku: svojstva se nakon točke navode kao ključna riječ bez navodnika, dok se unutar uglatih zagrada može navesti **izraz**! To znači da se on prvo izvršava, te se rezultat toga izvršavanja koristi kao "adresa" za pristupanje svojstvu. Dakle pomoću uglatih je zagrada omogućen *dinamički* dohvat svojstava.

Dakle ukoliko unaprijed znamo ime svojstva koji nam je potreban lakše je pisati `polje.length` nego `polje["length"]` . Međutim ako ne znamo unaprijed ime svojstva nego se ono određuje tijekom izvođenja programa, onda koristimo uglate zagrade. Dodatno, svojstvo može sadržavati i nedozvoljene znakove za imenovanje varijabli. U tom slučaju također koristimo uglate zagrade. Primjer:

```
let varijabla = {};
varijabla["svojstvo s razmakom"] = "tu sam";
varijabla.svojstvo s razmakom // Greška!
```

Metode

Svojstva varijabli (aka. "ono nakon točke") koja su po tipu **funkcije**, možemo još po uzoru na objektno-orijentirano programiranje nazvati i *metodama* . To su funkcije koje imaju pristup svim ostalim svojstvima varijable nad kojom su pozvane (aka. "ono ispred točke"). Primjer nad varijablom tipa `string` :

```
let a = "Neki string";

console.log(typeof primjer.toUpperCase); // kojeg tipa je svojstvo `toUppercase` u varijabli `a`? → "function"

console.log(primjer.toUpperCase()); // → "NEKI STRING"
```

> U ovom primjeru kažemo da je `toUpperCase` metoda `string` -a

Ranije spomenuto polje (`array`) nije od pretjerane koristi bez metoda koje služe za dodavanje novih elemenata:

```
let polje = [1, 2, 3];
polje.push(4);
polje.push(5);
console.log(polje); // → [1, 2, 3, 4, 5]
console.log(polje.pop()); // → 5
console.log(polje);// → [1, 2, 3, 4]
```

> Metoda `Array.pop()` služi za skidanje posljednjeg elementa s liste, dok metoda `Array.push(element)` dodaje element na kraj liste. Samim time implementacija liste ima sve potrebne metode za realizaciju strukture stoga (engl. *stack*).

Objekti

Svojstva i metode definirane nad jednostavnijim tipovima i nemaju previše koristi. Često čak i polje nije dovoljno da vjerno preslika podatke iz stvarnosti. Uzmimo za primjer da je potrebno spremiti u određenu strukturu zapis o pojedinim studentima. Rješenje s poljem i nije baš poželjno:

```
let student = ["Marko", "Horvat", "0303819345", "M"];
```

> Koje poteškoće možeš predvidjeti s ovakvim načinom zapisa?

Kada je potrebno pohraniti objekte koji sadrže određene imenovane atribute (npr. *ime* , *prezime* , ...) koristimo JavaScript `Object` tip podataka. Objekt je skup proizvoljnih atributa. Jedan od načina za kreiranje prethonog zapisa o studentu je sljedeći:

```

// definiranje objekta

let student = {
  ime: "Marko",
```

```
    prezime: "Horvat",
    JMBAG: "0303819345",
    aktivni_kolegiji: ["Programsko inženjerstvo", "Web aplikacije"],
    spol: "M",
    upisan: false
  };

  // pristup upravo definiranom objektu

  console.log(student.ime); // → "Marko"
  console.log(student.prezime); // → "Horvat"
  console.log(student.aktivni_kolegiji); // → ["Programsko inženjerstvo", "Web
  aplikacije"]
  student.upisan = true; // ispravak atributa
  console.log(student.upisan); // → true
```

> 💡 Primjeti kako atribut po tipu može biti bilo koji drugi JavaScript tip, tako i novi `object`, `array` ili `function`.

> ⚠️ Primjeti kako se `{}` koriste za definiranje blokova i za objekata! Prisjetimo se, blokovi služe sa određivanje vidljivosti mapiranja (varijabli).

Za razliku od objektno-orijentiranih jezika gdje se svojstva objekata definiraju klasama, ovdje vidimo da ne postoji koncept klase, već da se objekti oblikuju svaki zasebno. Samim time možemo im naknadno brisati ili dodavati nova svojstva. Primjer:

```
  let profesor = {
      ime: "Marko",
      prezime: "Marković",
      titula: "dr. sc."
  }

  delete profesor.titula;
  console.log(profesor); // nema više titule :(

  profesor.obrazovanje = "Doktor znanosti";
  console.log(profesor); // dodano je novo svojstvo
```

Ukoliko je potrebno dobiti uvid u svojstva pojedinog objekta, možemo koristiti metodu `keys` u ugrađenom objektu `Object`:

```
  // nastavak na prethodni primjer
  console.log(Object.keys(profesor)); // → [ 'ime', 'prezime', 'obrazovanje' ]
```

Analogno ukoliko želimo izlistati sva svojstva određenog objekta koristimo metodu `Object.values()`:

```
  // nastavak na prethodni primjer
  console.log(Object.values(profesor)); // → [ 'Marko', 'Marković', 'dr. sc.' ]
```

Možemo čak i kopirati svojstva određenog objekta na drugi objekt:

```
let student = {
    ime: "Ivica",
    prezime: "Ivić",
    status: "Apsolvent"
}
let profesor = {
    ime: "Marko",
    prezime: "Horvat",
    titula: "dr. sc."
}

Object.assign(student, profesor);
console.log(student); // prepisana su svojstva koja su već postojala...
```

⚠️ Ukoliko postoji isti naziv svojstva u odredišnom objektu, `Object.assign()` prepisuje vrijednost tog svojstva iz izvorišnog objekta (što je i očekivano ponašanje).

Veoma često u programiranju aplikacija u JavaScriptu imamo potrebu sačuvati kolekciju objekata. Primjerice, listu studenata koju je potrebno prikazati na zaslonu aplikacije. U tu svrhu možemo kombinirati tip **liste** i **objekta**:

```
let popis = [
    {
        ime: "Marko",
        prezime: "Marković"
    },
    {
        ime: "Iva",
        prezime: "Ivić"
    },
    {
        ime: "Lucija",
        prezime: "Lucić"
    },
    {
        ime: "Nikola",
        prezime: "Nikolić"
    }
];

console.log(popis); // → vraća listu objekata
console.log(popis[0].ime); // → ime prvog objekta u listi
console.log(popis[3].prezime); // → prezime četvrtog objekta

```

⚠️ Što će vratiti `popis[10]`, a što `popis[10].prezime`?

Drugi česti primjer jesu ugnježđeni objekti. U prethodnim primjerima smo susretali samo jednostavnije tipove podataka kao sastavnice objekata. Međutim svojstvo ili atribut objekta može biti bilo koji drugi tip podataka, uključujući i novi objekt ili polje. Generalno nema limita koliko duboko mogu objekti biti ugnježđeni. Limiti su praktične prirode te se obično u projektima koristi svega nekoliko razina ugnježđenosti. Ugnježđene objekte najčešće koristimo za realizaciju **kompozicije** objekata:

```
let student = {
    ime: "Lea",
    prezime: "Lovrinčić",
    upisan_studij: {
        naziv: "Informatika",
        sifra_ustanove: 234345
    },
    upisane_sifre_kolegija:  [1234, 9832, 329],
    pismeni_ispiti: [
        {
            datum: "12.9.2021",
            sifra_kolegija: 324,
            ocjena: 4
        },
        {
            datum: "12.2.2021",
            sifra_kolegija: 5653,
            ocjena: 5
        }
    ]
}

console.log("Šifra ustanove: " + student.upisan_studij.sifra_ustanove);
console.log("Ocjena s prvog pismenog ispita: " +
student.pismeni_ispiti[0].ocjena);
```

> Možeš li prepoznati kompoziciju i agregaciju u prethodnom primjeru? Možeš li razlikovati vezu "na više" od veze "jedan na jedan"? Kako se one općenito realiziraju?

Zadaci za vježbanje

1. Istraži jeli moguće rekurzivno postaviti objekt kao svojstvo tog istog objekta.

   ```
   let a = {
     naziv: "neki objekt"
   };
   a.unutarnji = a;
   console.log(a.unutarnji.unutarnji.unutarnji.naziv);
   ```

2. **(JS-401)** Sastavi listu od 10 studenata sa sljedećim svojstvima: `ime`, `prezime`, `upisan` (Boolean `true` / `false`). Vrijednosti svojstava proizvoljno odaberi. Sastavi funkciju `provjeri(lista, pojam)` koja vraća `true` ukoliko postoji student na `lista` čije ime ili prezime je baš `pojam`, a upisan je.

3. **(JS-402)** Nadogradi prethodni zadatak na način da ime i prezime ne moraju biti istovjetni pojmu, već je dovoljno da taj pojam sadržavaju. Neka pretraga bude neosjetljiva na velika i mala slova. Dodaj u tu funkciju još jedan parametar `status` na način da funkcija provjerava jeli `student.status` istovjetan primljenom parametru `status`. Drugim riječima, ne provjerava samo upisane studente nego se može specificirati status upisa.

4. **(JS-403)** Napiši funkciju `zagrade` koja će provjeriti jesu li zagrade valjano ugnježđene:

```
let zagrade = function(s) {
     // ... implementiraj me :)
};

console.log(zagrade("[()]()()")); // → true
console.log(zagrade("{[(((())))]}")); // → true
console.log(zagrade("({)}")); // → false
```

Hint: stog može pomoći :)

5. **(JS-404)** Implementiraj funkciju za još napredniju pretragu koja prima pojam koji može sadržavati više riječi odvojenih razmakom. Funkcija traži indeks **prvog** studenta u listi koji zadovoljava sve tražene pojmove bilo imenom, prezimenom, gradom ili njihovom kombinacijom. Implementiraj traženu funkciju `napredna_pretraga(pojam)` na način da prođu testovi (koristi se metoda `console.assert` koja u slučaju nejednakosti baca grešku):

```
let studenti = [
    {
        ime: "Marko",
        prezime: "Marković",
        grad: "Pula"
    },
    {
        ime: "Iva",
        prezime: "Ivić",
        grad: "Našice"
    },
    {
        ime: "Lucija",
        prezime: "Lucić",
        grad: "Zagreb"
    },
    {
        ime: "Nikola",
        prezime: "Nikolić",
        grad: "Rijeka"
    }
];

function napredna_pretraga(lista, pojam) {
    // ... implementirati ovdje :)
}

console.assert(napredna_pretraga(studenti, "ma ić") == 0) // → prvi student
```

```
29  console.assert(napredna_pretraga(studenti, "ko lić ri") == 3) // → zadnji student
30  console.assert(napredna_pretraga(studenti, "ić za") == 2) // → treći student
31  console.assert(napredna_pretraga(studenti, "ić ko la ri") == 3) // → zadnji
    student
```

6. **(JS-405)** Osmisli i oblikuj objekt po vlastitom odabiru koji sadrži barem jednu agregaciju i kompoziciju, te veze *na jedan* i *na više* .