**RAČUNARSKA GIMNAZIJA**

**MATURSKI RAD**

**iz predmeta**

**Napredne tehnike programiranja**

**„Vizualizacija algoritama za sortiranje“**

**Učenik Mentor**

**Lav Leon Hudak, dr Filip Marić**

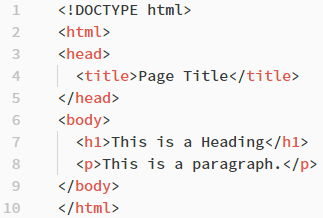
**Beograd, jun 2019.**

# Tehnologije korišćene u projektu

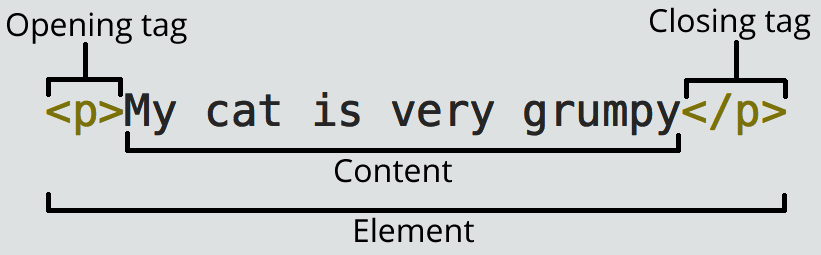
HTML

Hypertext Markup Language (HTML) je jezik koji se koristi za kreiranje veb stranica. Često se koristi u tandemu sa drugim jezicima, kao što su Cascading Stylesheets (CSS), JavaScript (JS) i PHP: Hypertext Processor (PHP).

HTML dokument je zapravo običan tekst fajl. Kada internet pretraživač otvori ovaj falj, on traži HTML kod i koristi ga da prikaže paragrafe, ubaci slike, napravi strukturu veb stranice itd. HTML fajl ima nekoliko osnovnih elemenata: *<html>, <head> i <body>*. Na primeru (1) može se primetiti da postoje i *</html>*, *</head>* i *</body>* tag-ovi. Ovo je zato što se skoro svi HTML elementi sastoje iz tag-a koji ih otvara, i tag-a koji ih zatvara. Takođe, može se videti da u fajlu postoji takozvana „parent-child“ struktura; elementi se stavljaju jedni unutar drugih.



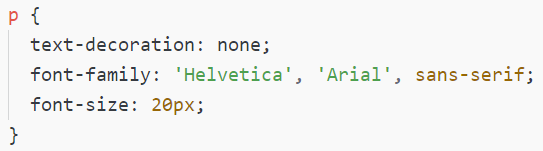
*(1) Primer osnovnog HTML koda*

**

*(2) Struktura HTML elementa*

CSS

Cascading Stylesheets (CSS) se koristi da „ulepšamo“ našu veb stranicu. Ako HTML daje strukturu veb stranici, CSS obezbeđuje pravila po kojima će se naši HTML elementi prikazivati. Možemo im menjati boju, veličinu, poziciju i mnogo više.



*(3) Primer CSS koda*

JavaScript

JavaScript (JS) je skripting jezik koji se koristi najviše kao dodatak veb stranicama, ali takodje ima upotrebu u drugim okruženjima kao što su Node.js i Apache. Koristeći JavaScript implementiramo stvari kao što su animacije, 2D i 3D grafike, interaktivne mape i ostale dinamičke delove naše stranice. JavaScript ima takozvanu „curly-bracket“ sintaksu, i spada u programske jezike visokog nivoa, što znači da je njegova sintaksa lako čitljiva za ljude.

Kako bismo uključili JavaScript falj u HTML stranicu, potrebno je dodati joj <script> tag, i dati mu atribut *src*. Ovom atributu prosleđujemo put do našeg JavaScript fajla, koji se završava ekstenzijom *.js*. <script> tag se može staviti bilo gde u stranici, ali pošto se stranica učitava odozgo na dole, kako bi se svi elementi učitali pre JS koda, dobra je praksa da se ovaj element stavi na sam kraj stranice.

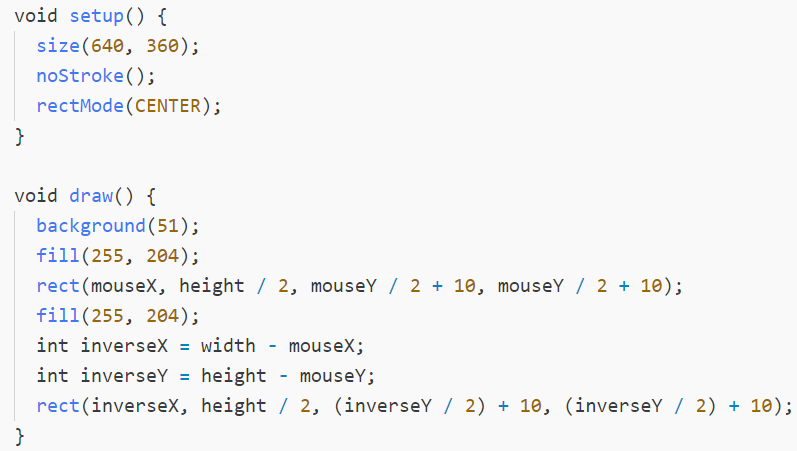


*(4) Primer JavaScript koda*

Processing

Processing je Java biblioteka otvorenog koda i okruženje koja omogućava pravljenje digitalne umetnosti, animacije i video igrica sa ciljem učenja ljudi o programiranju u vizuelnom kontekstu. Processing se prvi put pojavio 2001. godine, kada su se Java Applet-i koristili za prikazivanje grafičkih radova na internetu. Ovaj programski okvir koristi takozvani „Sketchbook“, na kome se iscrtava grafika. Pored mnogih funkcija koje omogućavaju lak grafički prikaz, Processing ima dve ključne funkcije:

1. **Setup** je funkcija koja se pokrene čim program započnesa radom. U njoj se definišu početne osobine programa, kao npr. veličina prozora u kome će se prikazivati. Svaki program može imati samo jednu setup() funkciju, i ona se ne bi trebalo pozivati sem automatski, na početku rada programa.
2. **Draw**, sa druge strane, je funkcija koja se izvršava beskonačno mnogo puta, dok se program ne zaustavi ili se pozove funkcija noLoop(). Poziva se automatski i posle njenog izvršavanja sadržina programa se ponovo iscrta na ekranu.



*(5) Primer Processing koda*

Processing ima širok spektar funkcionalnosti, kao što su:

* praćenje događaja kao što su *mousePressed* i *keyPressed*,
* crtanje fundamentalnih oblika kao što su kvadrati, trouglovi i poligoni,
* odeljak za matematiku; funkcije kao što su sinus i kosinus, interpolacija i sl,
* iscrtavanje Bezjeovih krivih, korišcenje Perlinovog šuma,
* pretvaranje polarnih koordinata u koordinate u Dekartovom koordinatnom sistemu,
* i mnogo više.

p5.js

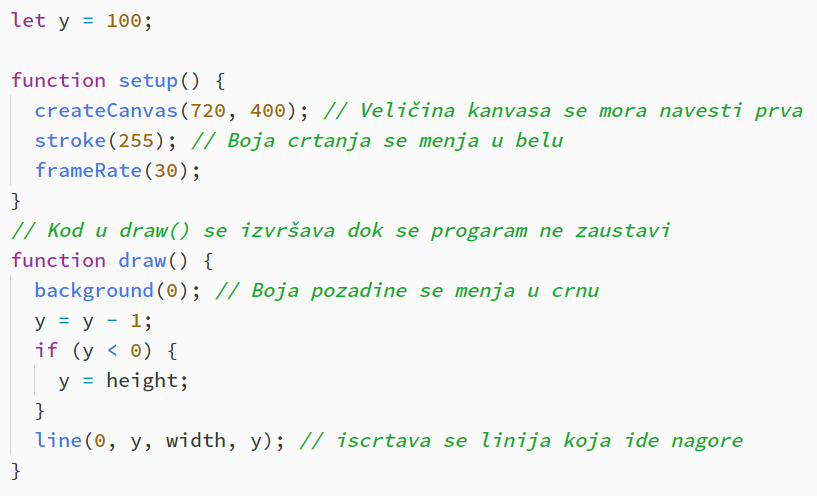
p5.js je JavaScript biblioteka napravljena na osnovu Processing-a. Napravljena je sa istim ciljem kao i Processing, tj. da približi svet programiranja dizajnerima, umetnicima i ljudima koji žele novi način da se kreativno izraze. p5.js koristi element <canvas> kao prostor na kome se iscrtava grafika, nalik Sketchbook-u u Processing-u. Ova biblioteka sadrži mnoge funkcije koje olakšavaju grafički rad, i neke od kategorija funkcija su:

* Color - sadrži funkcije za rad sa bojama, kao što je menjanje režima rada iz RGB u HSB,
* Shape - sadrži funkcije za crtanje primitivnih oblika, kao što su *circle()*, *rect()* i *point()*,
* Math - funkcije koje se bave vektorima i njihovim osobinama,
* Events - skup funkcija za rad sa događajima, kao što su *mouseMoved()* i *keyPressed()*.

p5.js takodje olakšava HTML aspekat programiranja, jer sadrži odeljak koji se bavi DOM-om. U ovom odeljku postoje funkcije koje se bave direktnim kreiranjem HTML elemenata iz JavaScripta. Primer ovakve funkcije je *createButton().* Takođe, postoje funkcije kao što su *child()* i *parent()*, koje omogućavaju pozicioniranje elemenata unutar HTML strukture.

Ako želimo koristiti p5.js biblioteku na našoj stranici, postoje dva načina:

1. možemo uključiti p5.js CDN link u naš <script> element, ili
2. možemo skinuti celu biblioteku (koja nije velika) u direktorijum našeg projekta.



*(6) Primer p5.js koda*

# Vizualizacija algoritama za sortiranje

Cilj projekta

Ako pogledamo oko sebe, možemo primetiti da smo okruženi nizovima različitih vrsta. Knjige na policama, kalendari, bilo kakvi natpisi, pa čak i slova na tastaturi. Ako ste ikada igrali tabliće ili poker, postoji šansa da ste karte u ruci poređali po boji, vrednosti ili nekom drugom redosledu, kako bi njihov odabir bio lakši. Kada želimo da rezervišemo sobu u hotelu, sajtovi nam pružaju opcije da prikažemo prvo one najjeftinije, najbliže centru ili sa najboljom ocenom. Još jedan primer su knjige u biblioteci, takođe poređane, i to leksikografski - po abecedi ili azbuci.

U svetu kompjutera takođe imamo nizove; to mogu biti nizovi brojeva, slova, slika i drugih objekata.

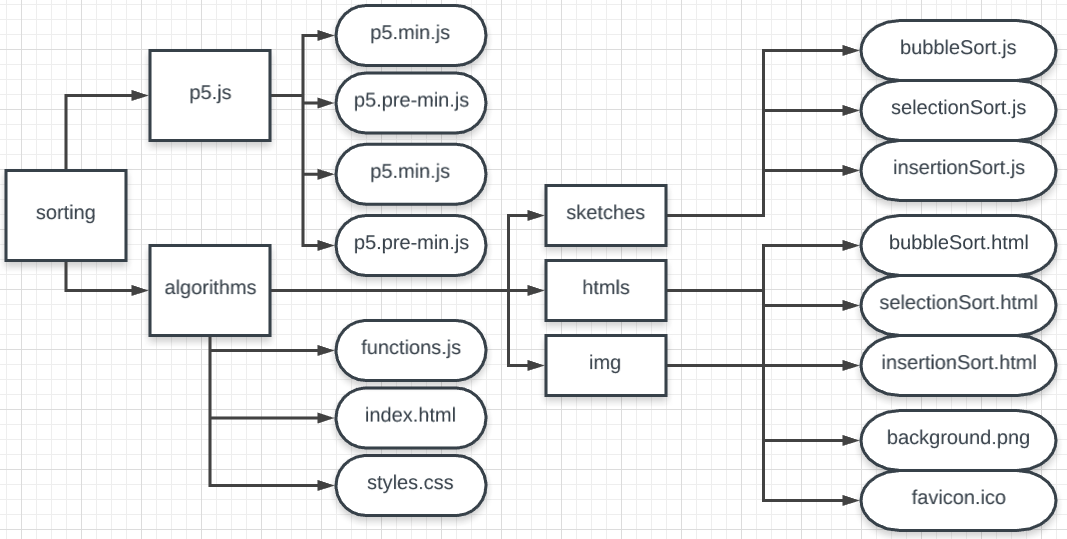
Struktura projekta

Svi fajlovi (izutev fajlova p5.js biblioteke) se nalaze u folderu *algorithms*. Glavna (početna) stranica projekta, *index.html*, sadži kratak uvod o tome šta su to nizovi u programiranju, par sintaksičkih primera, i kartice koje vode na ostale stranice. Ova stranica, kao i ostale HTML stranice, dele jedan CSS fajl, *styles.css*. Nakon toga, imamo tri direktorijuma:

* htmls,
* sketches,
* img.

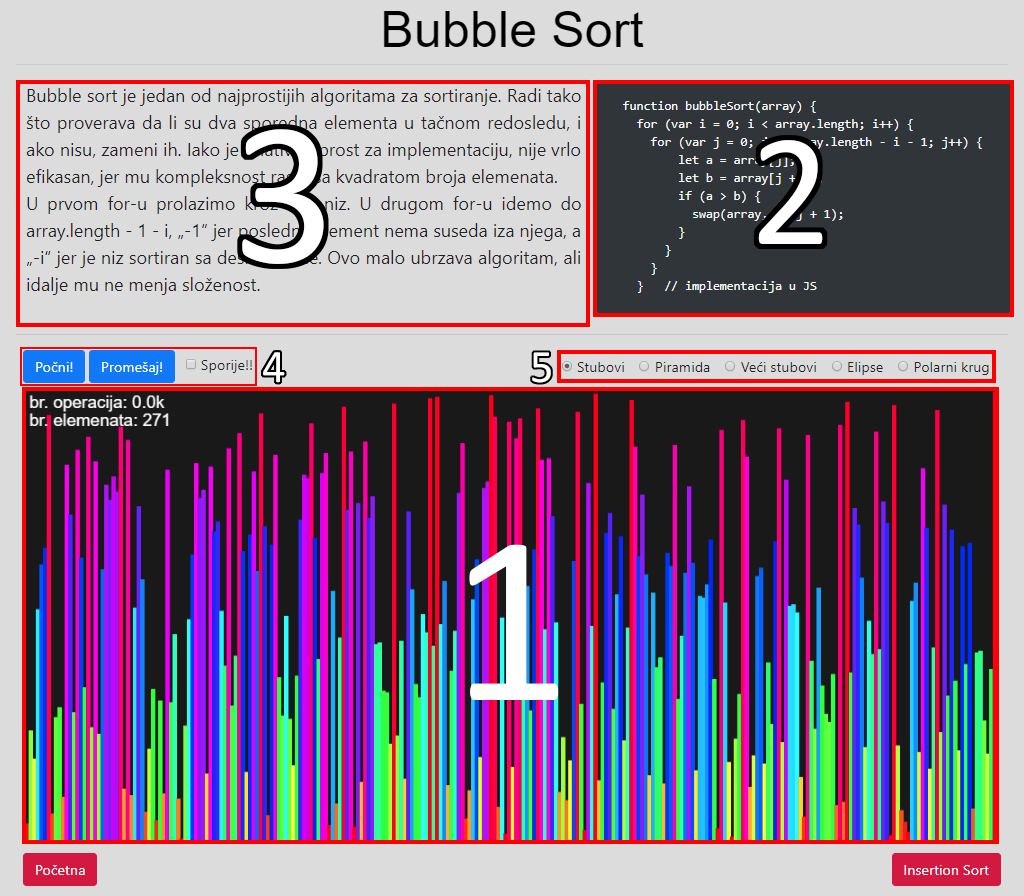
U direktorijumu *htmls* se nalaze HTML stranice za pojedinačne algoritme, naime Bubble Sort, Insertion Sort i Selection Sort. U direktorijumu *img* se nalaze pozadina, koju sve stranice koriste, i ikonica za sajt, a u direktorijumu *sketches* se nalaze pojedinačni JS fajlovi: *bubbleSort.js*, *insertionSort.js* i *selectionSort.js*. Pošto stranice koje prikazuju algoritme međusobno koriste dosta istog JS koda, ideja je da svaka od tih stranica sadrži fajlove *functions.js* (u kome se nalaze zajedničke funkcije i promenljive) i *xzySort.js* (koji implementira unikatni algoritam). Cilj ovoga je da se ponavljanje koda smanji na minimum, što olakšava navigaciju kroz sam kod.

U ovom projektu korišćen je popularni sistem za kontrolu verzija, *Git*, radi lakšeg premeštanja i očuvanja koda i njegove istorije. Git je povezan sa sajtom GitHub koji, uz nalog, daje veliku funkcionalnost za menadžment projekta, kao što su push i pull zahtevi, nove grane, i onlajn prikaz koda.



*(2) Struktura projekta*

Objašnjenje HTML stranice



*(1) Prikaz stranice bubbleSort.html*

1. Kanvas na kome je prikazan izmešani niz i informacije o radu algoritma
2. Kod algoritma koji sortira niz
3. Objašnjenje algoritma i njegovog koda
4. Dugmići za početak/zaustavljanje sortiranja, dugme za ponovno mešanje niza, checkBox koji omogućava sporije sortiranje niza
5. Dugmići za odabir načina iscrtavanja; trenutko prikazani *stubovi*

Sve stranice koje su zadužene za pojedine algoritme imaju isti raspored, ali drugačiji sadržaj.

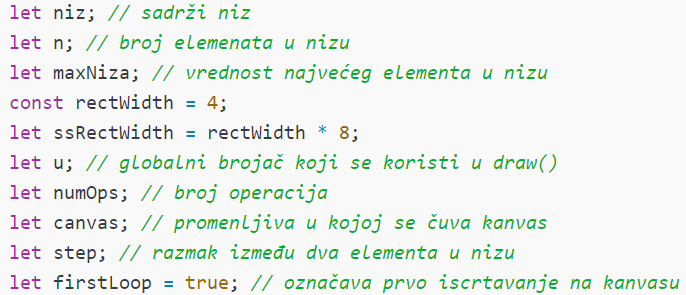
functions.js

U fajlu functions.js je kod koji dele sve tri stranice koje prikazuju algoritme. U njemu se nalaze:

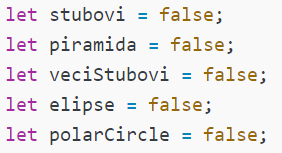
1. deklaracije globalnih promenljivih,
2. funkcija za resetovanje kanvasa i popunjavanje niza,
3. funkcije za rad sa nizom, kao što su *swap* i *shuffleArray,*
4. funkcije koje obrađuju događaje sa HTML stranice,
5. funkcije koje služe za iscrtavanje na kanvasu.

Globalne promenljive

Većina promenljivih korišćenih u ovom projektu su zajedničke za sve algoritme:

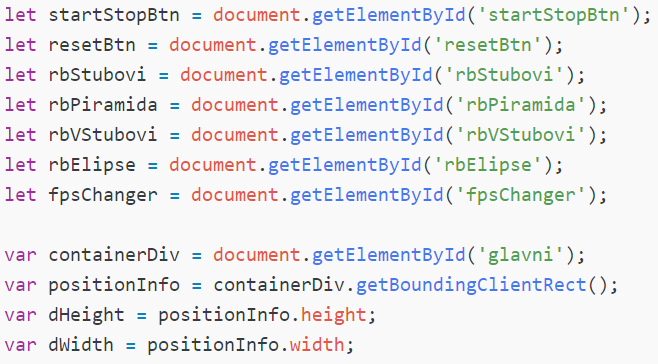


Takođe, imamo boolean promenljive koje određuju kako će se algoritam iscrtavati:



Ove promenljive se koriste i za pravljenje niza, jer je za svako iscrtavanje potreban malo drugačiji niz. Ovo se dešava u funkciji *resetSketch()*.

U sledećem bloku koda uzimamo podatke o HTML elementima na stranici, kao što su dugmići, checkBox-ovi i radioButton-i. Takođe uzimamo podatke kao što su širina i visina <div> elementa u kome se nalazi kanvas, na osnovu kojih kasnije određujemo veličinu i poziciju samog kanvasa.

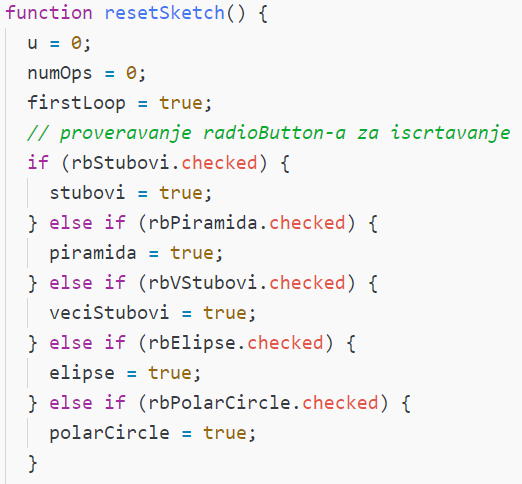


resetSketch()

Ova funkcija se poziva svaki put kada bi canvas trebao da se resetuje. Ove situacije su:

1. kada se pozove funkcija *setup()* - zato što je u *resetSketch()* funkciju prebačen kod koji se izvršava kada se prvi put niz pravi i izcrtava na kanvasu,
2. kada korisnik klikne dugme „Promešaj!“,
3. kada se veličina prozora promeni,
4. u funkciji *rbChanged()*, koja je obrađivač događaja za radioButton-e koji menjaju način iscrtavanja na kanvasu.

*resetSketch()* ima posao da vrati sve na početno stanje, a potom proveri kakvo iscrtavanje je izabrano, i napravi niz u odnosu na to.



Funkcija prvo vraća globalni brojač *u* i broj operacija na vrednost 0, a zatim vrednost promenljive *firstLoop* na true. Nakon toga, proverava koji način iscrtavanja je izabran, i na osnovu toga daje vrednost true jednoj od promenljivih u odnosu na koju se kasnije pravi niz.

Sledeći deo koda se bavi određivanjem broja elemenata i njihovih vrednosti u nizu. Niz koji nastaje je aritmetički niz, u kome je razlika između dva elementa vrednost promenljive *step*. p5.js nam daje promenljive *width* i *height,* čije su vrednosti širina i visina kanvasa.



U slučaju **piramide**, imamo elemenata koliko i piksela po širini kanvasa. Svaka linija je zapravo element. Visinu, tj. polovinu visine ove linije daje nam vrednost elementa. Kako bismo imali linije od sredine leve ivice do gornjeg i donjeg desnog ugla kanvasa, potrebno je da *step* bude 1/2 količnika visine kanvasa i dužine niza.[[1]](#footnote-1)

U slučaju **stubova** i **većih stubova**, broj elemenata se računa kao *ceo[[2]](#footnote-2)* broj najbliži odnosu širine kanvasa i stubova koji se iscrtavaju. Kako bismo imali „liniju“ od donjeg levog ugla do gornjeg desnog ugla kanvasa, *step* se računa kao količnik visine kanvasa i dužine niza.

Za **elipsu** su nam potrebne vrednosti velike i male poluose; njih dobijamo preko pozicije elementa u nizu. Ovo znači da će širina elipse biti širine kanvasa, a pola od toga.

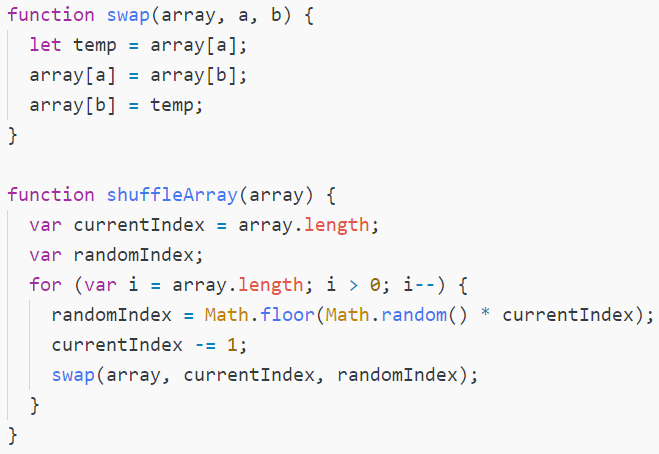
Za **polarni krug i**mamo 360 elemenata, i fiksiranu vrednost promenljive *step.*

Potom popunimo niz prostim *for* ciklusom, i sačuvamo najveću vrednost u nizu pre nego što ga promešamo.

Kako bismo prikazali spektar boja, koristimo funkciju *colorMode()* da izaberemo režim rada HSB („Hue, saturation, brightness“), kojoj kao drugi agrument treba proslediti *range,* tj. maksimalnu vrednost. Da bismo prikazali ceo spektar u tačnom odnosu sa načinom iscrtavanja, ovde prosleđujemo promenljivu *maxNiza.*

Funkcije za rad sa nizom

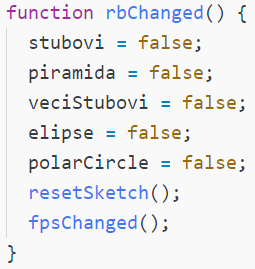
Imamo dve funkcije za rad sa nizovma, *swap()* i *shuffleArray().*

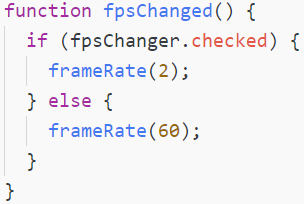


**swap()**uzima kao argumente niz na kome radi, i pozicije elemenata koje treba da zameni. Potom sprovodi prostu zamenu elemenata koristeći privremenu promenljivu.

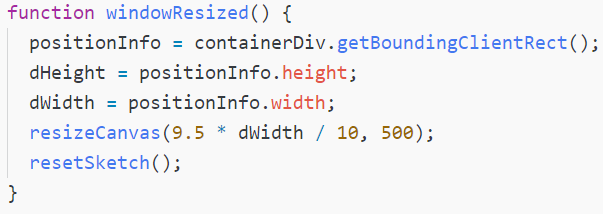
**shuffleArray()** uzima niz koji treba da izmeša kao argument. Kao početnu vrednost uzimamo broj elemenata u nizu. Kako bismo dobili neku nasumičnu poziciju u nizu, označenu sa *randomIndex*, uzimamo vrednost *currentIndex* i množimo je sa Math.random(), koja vraća brojeve od 0 do 1, isključujući 1, a zatim je zaokružujemo na najbliži ceo broj ispod. Potom oduzimamo jedan od *currentIndex* i pozivamo funkciju *swap()*, koja zamenjuje elemente na poziciji *currentIndex* i *randomIndex*. Ovo se ponavlja puta koliko imamo elemenata u nizu, kako bismo bili sigurni da je ceo niz promešan.

Funkcije koje obrađuju događaje sa HTML stranice





Funkcija *rbChanged()* je obrađivač događaja za sve radioButton-e koje su na stranici. Ova funkcija resetuje sve boolean vrednosti za crtanje na false, poziva *resetSketch()* (u kojoj se proverava koji način iscrtavanja je izabran), i poziva funkciju *fpsChanged()*, koja je zadužena da obradi događaj od checkBox-a koji menja broj kadrova koji se iscrtavaju na kanvasu po sekundi.



Funkcija *windowResized()* je ugrađena u p5.js biblioteku i poziva se kada se veličina prozora u kome se kanvas prikazuje promeni. Uzima se nova visina i širina <div> elementa u kome se nalazi kanvas, poziva se ugrađena funkcija *resizeCanvas()* koja menja širinu kanvasa na 95% <div> elementa u kome je, a potom se poziva funkcija *resetSketch()*.



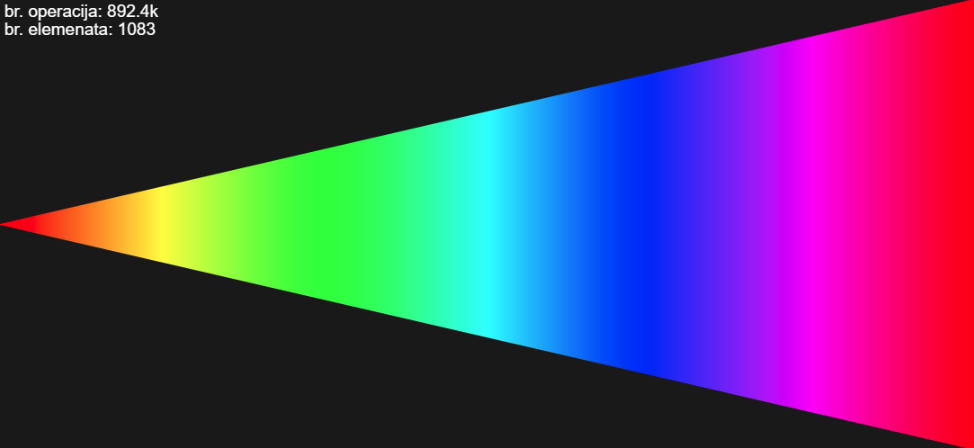
Funkcija *startStopSketch()* je zadužena da pauzira i pokrene iscrtavanje na kanvasu. Ona se poziva na događaj *onClick* dugmeta *startStopBtn*. Ovde koristimo funkcije *loop()* i *noLoop()*, koje su ugrađene u p5.js biblioteku. Ovo su funkcije koje zapravo prekidaju u ponovo poreću iscrtavanje na kanvasu. Pored toga, menjamo vrednost firstLoop na false (jer više nije prvo iscrtavanje), i menjamo tekst na dugmetu kako bi se uklapao sa stanjem kanvasa.

Funkcije za iscrtavanje na kanvasu

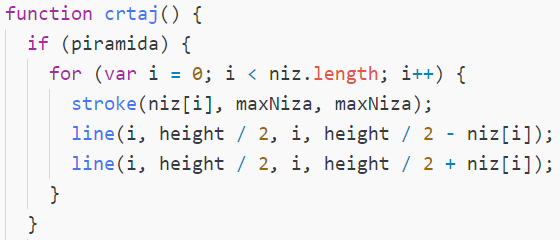
Funkcija *crtanje()* ima *if else* slučajeve od kojih se svaki bavi iscrtavanjem posebnog oblika. To su:

1. piramida,
2. stubovi i veći stubovi,
3. elipse,
4. polarni krug.

**Piramida**

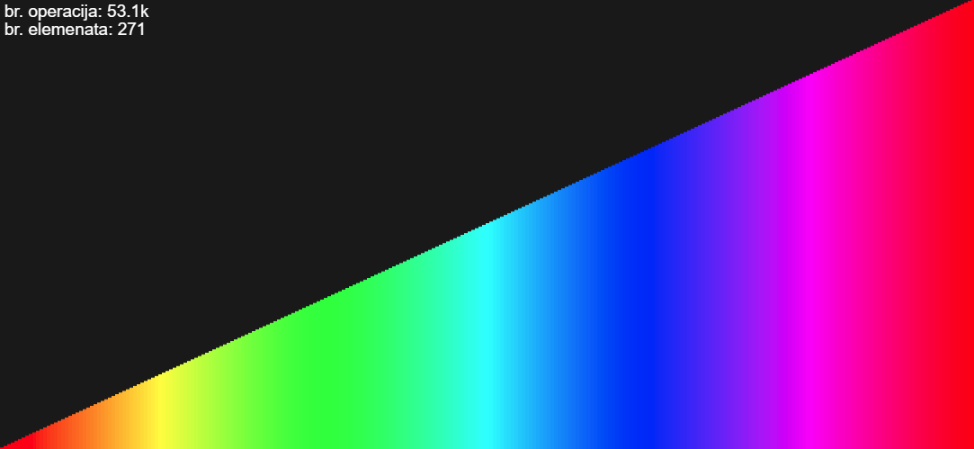


Kako bismo iscrtali spektar boja, koristimo funkciju *stroke()* koja menja boju ivice crtanja. *stroke()* se može pozivati sa različitim brojem argumenata, naime jedan, tri ili četiri. U našem slučaju, poziva se sa tri argumenta. Ako bismo bili u RGB režimu rada, ovo bi bili parametri *red*, *green* i *blue,* ali pošto koristimo HSB režim rada, ovo su *hue*, *saturation* i *brightness*. Pošto smo u funkciji *resetSketch()* prosledili vrednost maxNiza kao maksimalne vrednosti ova tri parametra, za *brightness* i *saturation* prosleđujemo maksimalnu vrednost, a za hue vrednost niza u tom trenutku, pošto nam je niz artitmetički i ravnomerno raspoređen. Ovaj način razmišljanja je isti za sve načine crtanja.

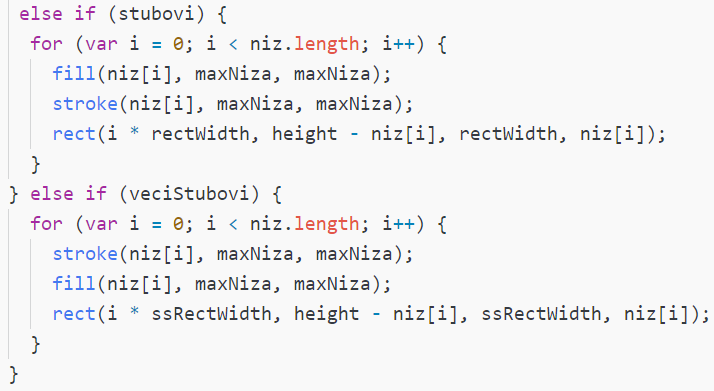


Kod piramide svaka linija (dve linije) predstavljaju jedan element. Piramida ima osu simetrije po liniji *height / 2*. Funkcija kojom crtamo linije, *line()*, uzima početnu tačku i krajnju tačku, tj. njihove koordinate. Prva tačka je ista za obe linije, i njena x koordinata je ustvari pozicija trenutnog elementa u nizu, a y koordinata se nalazi na osi simetrije. Druga tačka je na istoj vertikali kao i prva, ali joj je y koordinata pomerena za vrednost niza u pozitivnom ili negativnom smeru.

**Stubovi i veći stubovi**

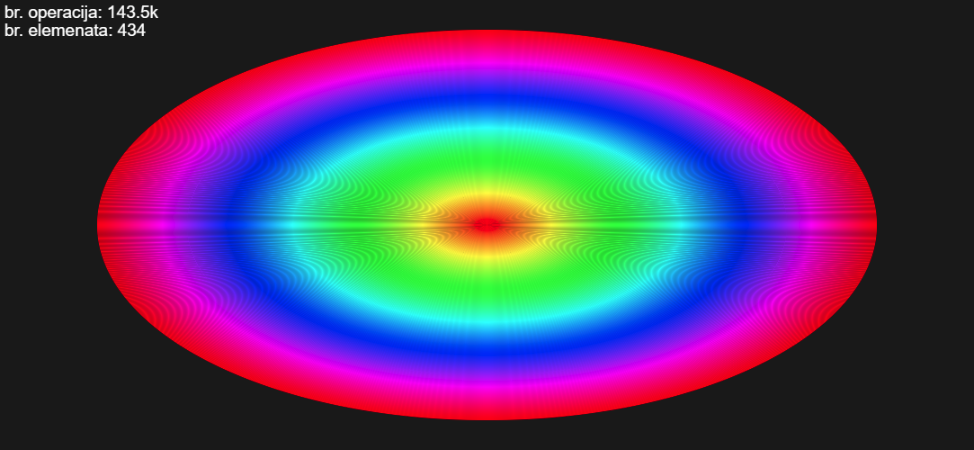


Jedina razlika kod stubova i većih stubova jeste zapravo širina stubova. Širina stubova je određena promenljivom *rectWidth*, a širina većih stubova je samo umnožak ove promenljive.

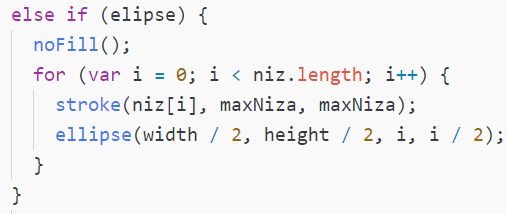


Funkcija *rect()* uzima gornju levu tačku, visinu i širinu pravougaonika kojeg crta. Pošto su stubovi jedan do drugog, *x* koordinata gornje leve tačke će biti proizvod širine stuba i njegove pozicije u nizu, a *y* koordinata se nalazi na poziciji *height* minus vrednost trenutnog elementa. Širina stuba je opet, promenljiva *rectWidth*, a visina stuba je vrednost elementa u tom trenutku. Sistem crtanja većih stubova je poptuno isti.

**Elipse**



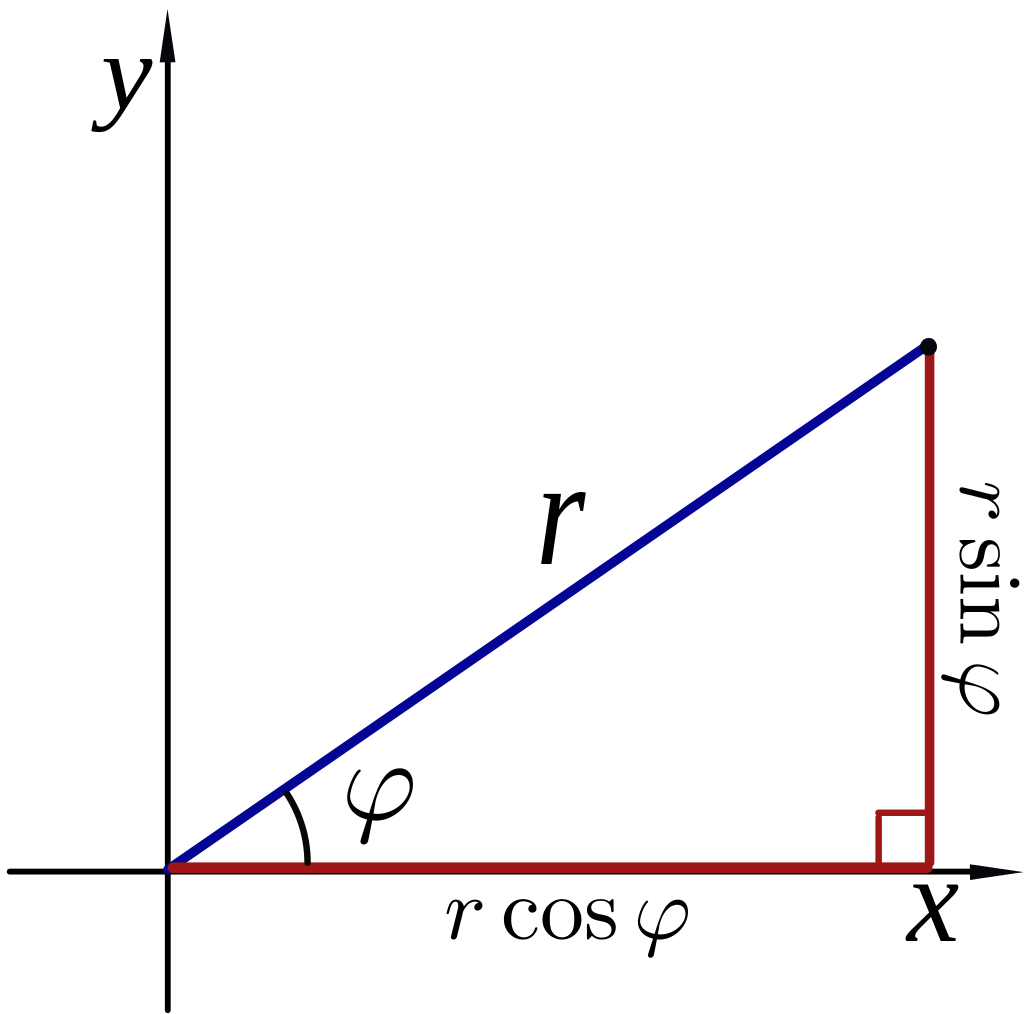
Ideja kod ovog prikaza jeste da se za svaki element nacrta jedna elipsa, koja nije popunjena. Kako elipse ne bi bile popunjene koristimo p5.js funkciju *noFill()*. Zatim, biramo boju za trenutnu elipsu, i crtamo je sa p5.js funkcijom *ellipse().* Ova funkcija uzima kao parametre centar elipse, njenu visinu i njenu širinu. Pošto se centar elipse nalazi u centru kanvasa, prosleđujemo joj koordinate centra *width / 2* i *height / 2*. Za širinu joj dajemo trenutnu poziciju elementa u nizu, a za visinu pola od toga.



**Polarni krug**



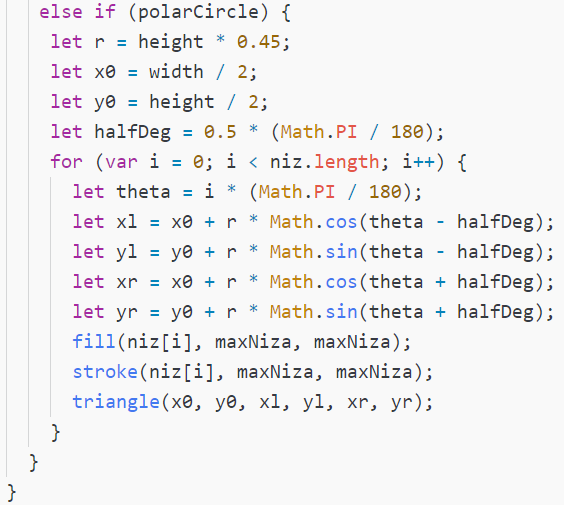
Za svaki element u nizu crta se poluprečnik ovog kruga. Pošto su nam elementi u nizu celi brojevi od 1 do 360, pozicija elementa u nizu je zapravo i ugao pod kojim će on biti crtan. Kako bismo odredili koordinate za crtanje, koristimo sistem polarnih koordinata, koji je objašnjen ispod.



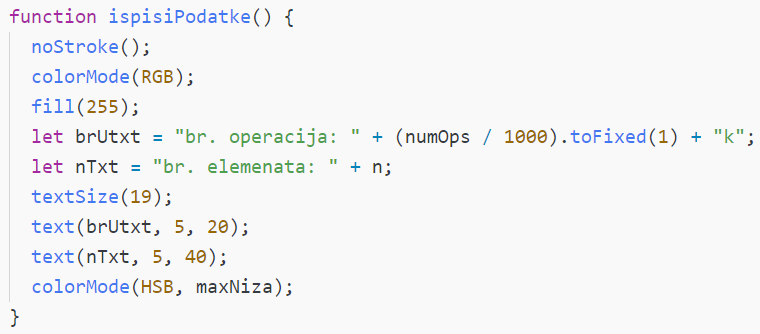
*(1) Sistem polarnih koordinata*

Svaka tačka na kružnici je određena sa uglom φ i razdaljinom od centra kruga *r. X* koordinata je data jednačinom *r \* cos(φ)*, a *y* koordinata jednačinom *r \* sin(φ)*.

Pošto funkcije Math.sin i Math.cos uzimaju ugao u radijanima, koristimo formulu navedenu dole da u promenljivu *theta* sačuvamo trenutni ugao pod kojim crtamo *trougao* (umesto linije). Crtamo trougao zbog lepšeg prikaza. Tačke na kružnici koje trougao sadrži su tačke koje su pomerene za pola stepena unazad, tj. unapred od ugla pod kojim se iscrtava trenutni element. Funkcija *triangle()* za argumente uzima koordinate sve tri tačke, a *fill()* i *stroke()* boje ovaj trougao.



**Funkcija za ispis podataka o algoritmu**



U funkciji *ispisiPodatke()* u levom gornjem uglu kanvasa ispisujemo podatke o broju elemenata u nizu koji se trenutno prikazuje, i broj operacija koje je algoritam izvršio do datog trenutka. Moramo promeniti režim rada u RGB jer se u odnosu na način iscrtavanja menja *range*, tj. maksimalna vrednost, pa ne bismo mogli da imamo belu boju sve vreme. Broj operacija se ispisuje u hiljadama tako što podelimo pravi broj operacija sa hiljadu i odsečemo sve osim poslednje decimale broja kojeg smo dobili sa funkcijom *toFixed()*. Nakon što ispišemo sve podatke, vraćamo režim rada na HSB.

1. Detaljnije objašnjeno iscrtavanje svih oblika u odelku „Funkcije za iscrtavanje na kanvasu“. [↑](#footnote-ref-1)
2. Broj elemenata u nizu mora biti ceo broj. [↑](#footnote-ref-2)