**Regular Expressions (REGEX)**

* Regex se koristi za proveravanje stringova tj. da li uneti string odgovara nekom patternu koji smo uneli u regex (npr. kod provere e-maila da li je npr unet @ ili da li ima neki odgovarajuci nastavak, itd.), ali stvara korist regexa nije da trazi neke stringove nego patterne koji u kombinaciji daju neke stringove.
* **REGEX JE CASE SENSITIVE!** (osim ako se ne postavi flag pa da se naveda da bude insensitive)
* Za testiranje moze se koristiti sajt [regex101.com](https://regex101.com/)

**Basic**

**/ pattern /**

* Svaka regex pocinje i zavrsava se sa **/.** Znaci regex se pise izmedju dve /.

**/ ninja /**

* Ako ukucamo obican string onda ce kao pattern da se i trazi ceo taj string (npr. ninja i onda ce da se trazi taj string u nekom stringu). **Ako** u stringu u kome trazimo ima i drugih karaktera/stringova onda ce svejedno funkcija da vrati true jer nismo naveli da tacno string mora da bude taj neki string nego da pronadje taj string bilo gde u stringu u kome se trazi. **Ovo** takodje vraca i prvu pojavu ovog stringa u tom nekom stringu tako da nece da trazi dalje.

**Flags**

* Regex flags omogucavaju da se omoguce neke dodatne mogucnosti. Flag se navodi nakon poslednje / sa odgovarajucim slovom za taj flag. Moguca je i kombinacija vise flagova (npr. **/ pattern / gi** )
* Tu spadaju: **global, multi line, insensitive, sticky, unicode i single line.**
* **Global –** Oznacava se sa **g.** Omogucava da se nadju sve pojave patterna u nekom stringu. Po defaultu regex vraca prvu pojavu patterna, ali sa ovim flagom vracaju se sve pojave tog patterna (npr. **/ ninja / g** )
* **Insensitive –** Oznacava se sa **i.** Omogucava da regex bude insensitive tj. da trazi pattern u stringu nezavisno od velikih i malih slova. Po defaultu regex je case sensitive. (npr. **/ ninja / i –** vraca i Ninja, NINJA, nInja, itd...)

**Pronalazenje navedenih karaktera u stringu – [ ]**

* Ako zelimo da pronadjemo sve stringove koji mogu da pocinju slovima npr. a ili b, a da im je ostatak karaktera isti onda se koriste **[]** zagrade. Na odgovarajucoj poziciji u stringu stavljamo uglaste zagrade I u njima sve karaktere koji tu mogu da stoje. Ovo pokriva samo jednu poziciju u stringu, tako da ako je vise karaktera u zagradi onda ce da se ispituje za svaki od njih.

**/ [nq]inja /**

* U ovom slucaju traze se stringovi koji pocinju sa n ili q i ostatak im je inja znaci ninja i qinja.

**/ [abc123] /**

* U ovom slucaju vraca se true ako se pronadje bilo koji od ovih karaktera u stringu i to bilo gde jer pre njega i posle njega nemamo nista vise tako da nije bitno gde se u stringu nalaze ovi karakteri.

**Pronalazenje svih karaktera osim navedenih u stringu - ^**

* Ideja je ista kao kod prethodnog slucaja samo ako imamo da na nekoj poziciji mozemo da imamo bilo koji karakter osim jednog ili dva, da ne bi navodili kao u prethodnom slucaju sve te karaktere koristi se karakter **^** koji oznacava negaciju u regexu. Svi karakteri nakon ovog znaka nece se traziti u stringu.

**/ [^a] /**

* U ovom slucaju vracaju se svi karakteri koji nisu a.

**/ [^0]007 /**

* U ovom slucaju vracaju se sve pojave kombinacije nesto 007 osim kombinacije 0007.

**Kako obuhvatiti samo neka slova – Range**

* Da u nekim patternima ne bi upisivali npr. sva slova abecede jednostavno se koristi range koji kaze od kog do kog karaktera moze da bude taj pattern. Pise se **char – char,** sa ovom crticom izmedju dva karaktera.

**/ [a-z]ing /**

* U ovom slucaju vracaju se svi stringovi koji pocinju sa bilo kojim slovom abecede (od a do z ali mala slova) I sa nastavkom ing. Ne mora da bude bas od a do z moze I npr. od b – g, itd.

**/ [^a-z]ing /**

* Ovaj slucaj je prethodni slucaj samo negacija, tj. validne su sve reci koje ne pocinju malim slovima od a-z, to znaci da su validne npr Aing, Bing, itd…

**/ [a-zA-Z]ing /**

* Ovde je isto kao u prethodnom slucaju sto se tice malih slova samo sto je dodat I range za velika slova, jer ako nismo stavili da nam je regex insensitive onda mora ovako da se stavi da bi se pronalazila I velika I mala slova.

**/ [0-9]ing /**

* Isto kao u prethodnim slucajevima samo sa brojevima.

**Uzastopna provera za svaki karakter – {}**

* Ako npr. hocemo da proverimo da li je korisnik uneo broj telefona od 11 cifara to mozemo da uradimo tako sto [0-9] napisemo 11 puta. Sa druge strane postoji operator + koji se stavi pored [0-9] I koji ako samo tako stoji kaze da ce ovaj pattern da se primeni na svaki karakter (isti efekat kao da nismo napisali +). Ako hocemo za telefon od 11 cifara onda pored [0-9] stavimo {11} sto kaze da ce da se proveri ovaj pattern da li je ispunjen za uzastopno 11 karaktera, ako barem jedan nije ispunjen onda se nista ne vraca.

**/ [0-9]{11} /**

* U ovom slucaju sve kombinacije 11 uzastopnih cifara vraca se da je pronasao (01234567890).

**/ [a-z]{5} /**

* U ovom slucaju vraca se bilo koja rec od 5 slova, jer je stavljen range od a – z (hello).

**/ [a-z]{5,8} /**

* Kao I prethodni slucaj samo sto ce ovaj pattern da se primeni na 5 do 8 uzastopnih karaktera, tako da ce sve reci koje su duge 5,6,7 ili 8 karaktera biti validne (hello, helloo, itd…).

**/ [a-z]{5,} /**

* Ovo je greedy slucaj gde se u obzir uzimaju sve reci koje su duzine 5 ili vise karaktera, tj. nisu validne samo reci koje imaju manje od 5 karaktera.

**Razdvajanje delova patterna – ()**

* Regex pattern mozemo podeliti na delove sa () ali to nikako ne utice na krajnji rezultat regex-a.

**/^([a-z\d\.-]+)@([a-z\d-]+)\.([a-z]{2,8})(\.[a-z]{2,8})?$/**

* Ovo je primer patterna za proveru unosa e-maila. Vidimo da sa zagradama je kao podeljen na 4 dela. Prvi deo je za naziv, pa onda mora da ide @, drugi deo je domem, nakon cega mora da ide tacka, treci deo ide nastavak I cetvrti deo je opcion zbog ? tj. da moze da bude, a ne mora.

**Metacharacters**

* Metakarakteri nam omogucavaju lakse pisanje regex-a. Meta karakteri se pisu sa **\** ispred njih jer bez toga su samo obicni karakteri. Postoje mnogi metakarakteri ali najvise se koriste: **\d, \w, \s, \t, itd…**
* **\d –** odnosi se na bilo koju cifru od 0 – 9. Isto kao da smo napisali [0-9]. Ako unesemo 123 vrati ce se 3 pogodka jer su I 1 I 2 I 3 cifre.
* **\D –** ovo je negacija na prethodni metakarakter \d, tj. odnosi se na sve karaktere koje nisu cifra (slova, ., , ,itd…)
* **\w –** odnosi se na bilo koji karakter od a – z, A – Z, ukljucujuci I cifre od 0 – 9 I donju crtu \_. Isto kao da smo napisali sve ovo samo u [].
* **\W –** ovo je negacija na prethodni metakarakter \w tj. odnosi se na sve karaktere koji nisu u skupu prethodnog karaktera (npr. @, ?, !, itd…)
* **\s –** odnosi se na prazne karaktere tipa spejs, tab, itd…
* **\S –** ovo je negacija na prethodni metakarakter tj. odnosi se na sve karaktere koji nisu prazni karakteri
* **\t –** odnosi se samo na tab karakter.

**/ \d\s\w /**

* Ova kombinacija kaze da prvi karakter reci mora da bude bilo koja cifra, drugi karakter spejs I treci karakter bilo koje slovo (veliko, malo), cifra ili \_ (5 n, 6 s).

**/ \d{3}\s\w /**

* Ova kombinacija je slicna prethodnoj samo u kombinaciji sa {} kaze da prva tri karaktera moraju biti cifra od 0 do 9, pa zatim neki spejs/tab I na kraju neko slovo Ili cifra (037 k).

**/ \d\s\w /**

* Ova kombinacija kaze da prvi karakter reci mora da bude bilo koja cifra, drugi karakter spejs I treci karakter bilo koje slovo (veliko, malo), cifra ili \_ (5 n, 6 s).

**Special characters**

* Najvise korisceni specijalni karakteri su: **+, \, [], [^], ?, ., \*, ^, $, itd…**
* **+ -** kada se on nadje pored karaktera to znaci da taj karakter treba da se pojavi barem jedanput ili vise puta npr. **a+** validni karakteri su a, aa, aaa, itd…
* **\ -** kada se on nadje ispred nekog karaktera dobijemo metakaratkere koji imaju neko svoje znacenje (pronaci deo o metakarakterima), ali takodje ako hocemo u stringu da trazimo neke specijalne karaktere kao sto su ., ?, !, itd… onda u paternu samo navedemo \ pa taj specijalni karakter I onda ce u stringu da se trazi I on kao obican karakter. Npr. **abc\. –** validno je samo abc. jer ovde . nema funkciju da je bilo koji karakter nego bas da je tacka.
* **[] –** kada imamo ove uglaste zagrade to znaci da je karakter validan ako I samo ako se nalazi u skupu koji je napisan unutar uglastih zagrada npr. **[a]** validne kombinacije sve reci koje imaju a u sebi.
* **[^] –** ovo je negacija na prethodni specijalni karakter, tako da je karakter validan ako I samo ako se ne nalazi u skupu koji je unutar uglastih zagrada npr. **[^a]** validne sve kombinacije reci koje nemaju a u sebi.
* **? –** kada se nadje posle karaktera znaci da taj karakter moze da se javi jednom ili nijednom (opciono) sto znaci da su validne sve reci koje imaju ili nemaju taj karakter na odredjenom mestu npr. **hello?** – validno je hello I hell (u prvom slucaju javlja se jednom o a u drugom se ne javlja tako da su oba validna), npr. **a[a-z]?** – validno je a, a I bilo koje drugo malo slovo od a-z (u prvom slucaju a je samo jer drugo slovo je opciono I zato je ovo validno)
* **. –** znaci da karakter moze da bude bilo sta osim newline karaktera. Npr. **car. –** validno je card, cars, carb, itd… ali nije validno samo car jer smo stavili da 4ti karakter moze da bude bilo sta I zato mora da se navede I 4ti karakter, npr. **.+ -** validan je bilo koji string sa barem jednim karakterom.
* **\* -** isto kao I + samo sto karakter moze da se ne pojavi ili da se pojavi vise puta. Npr. **a[a-z]\* -** validno je sve od a do a pa bilo koje malo slovo jer \* kaze da drugi karakter moze da bude od a-z ili ne mora da bude.
* **^ -** ako se nadje ispred nekog karaktera to znaci da je karakter validan ako se nalazi na pocetku stringa. Npr. **^a** validni su sve reci koje pocinju sa a, **^[a-z]{5}** validna je prvih 5 karaktera u stringu koje su od a-z.
* **$ -** ako se nadje iza nekog karaktera to znaci da je taj karakter validan ako se nalazi na kraju stringa. Npr. **a$** validne su sve reci koje se zavrsavaju sa a, **[a-z]{5}$** validna je rec od 5 karaktera koja se nalazi na kraju stringa a da ima karaktere od a-z.

**/ ^[a-z]{5}$ /**

* U ovom slucaju znaci da rec mora da bude od 5 karaktera tj. da ce biti validno samo ako se unese rec od 5 karaktera, ako je karaktera manje ili vise od 5 onda regex ne vraca pogodak. Npr. validno je ninja, ali nije ninjas, ninjaaa, itd…
* Kombinacija **^ I $** se najvise koristi kod forma kada se zeli da se proveri da li je uneto tacno neki broj karaktera.

**Alternate characters**

* **| -** ovaj znak je OR u regexu. Koristimo kada zelimo da proverimo da li je jedan ili drugi karakter.

**/ p|t /**

* U ovom slucaju validno je ako se napise ili p ili t.

**/ (p|t)yre /**

* U ovom slucaju validno je ako se napise ili p ili t, a da je nastavak yre, znaci pyre ili tyre.

**/ .+@(gmail|yahoo|hotmail)\.com /**

* Ovo je slucaj za proveru unete e-mail adrese. Kaze da pre @ moze da bude bilo koji broj karaktera, a da posle @ moze da bude gmail, yahoo ili hotmail uz .com.

**/ (p|t)?yre /**

* U ovom slucaju validno je ako se napise ili p ili t uz nastavak yre ili da se uopste ne napisu zbog specijalnog karaktera ?, tako da je validno pyre, tyre i yre.

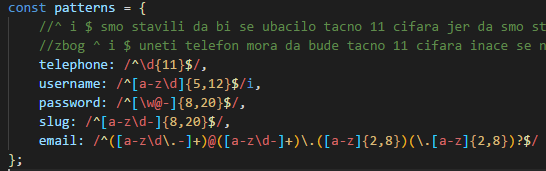
**Regex in JavaScript**

* Regex se u JavaScrip pisu ne kao string izmedju „“ nego izmedju / / uz to da posle druge / mozemo da stavimo flagove koji nam trebaju g,i, itd... Regex mozemo da pamtimo u neku promenljivu i samo da koristimo kad nam treba. Imamo dva nacina da napisemo Regex u JavaScript:





* Dobra ideja za cuvanje regex patterna je kao objekat koji u svakom propertiju ima neki pattern koji nam treba.



* Funkcija koja se koristi za testiranje je **test** koja je deo regex promenljive koja sadrzi pattern, a kao parametar joj prosledjujemo vrednost koju ispitujemo.



* Ovi patterni se najcesce koriste da kada korisnik unese neki karakter on se odmah proveri sa ovim patternom. To se radi sa sledecim JS kodom:

