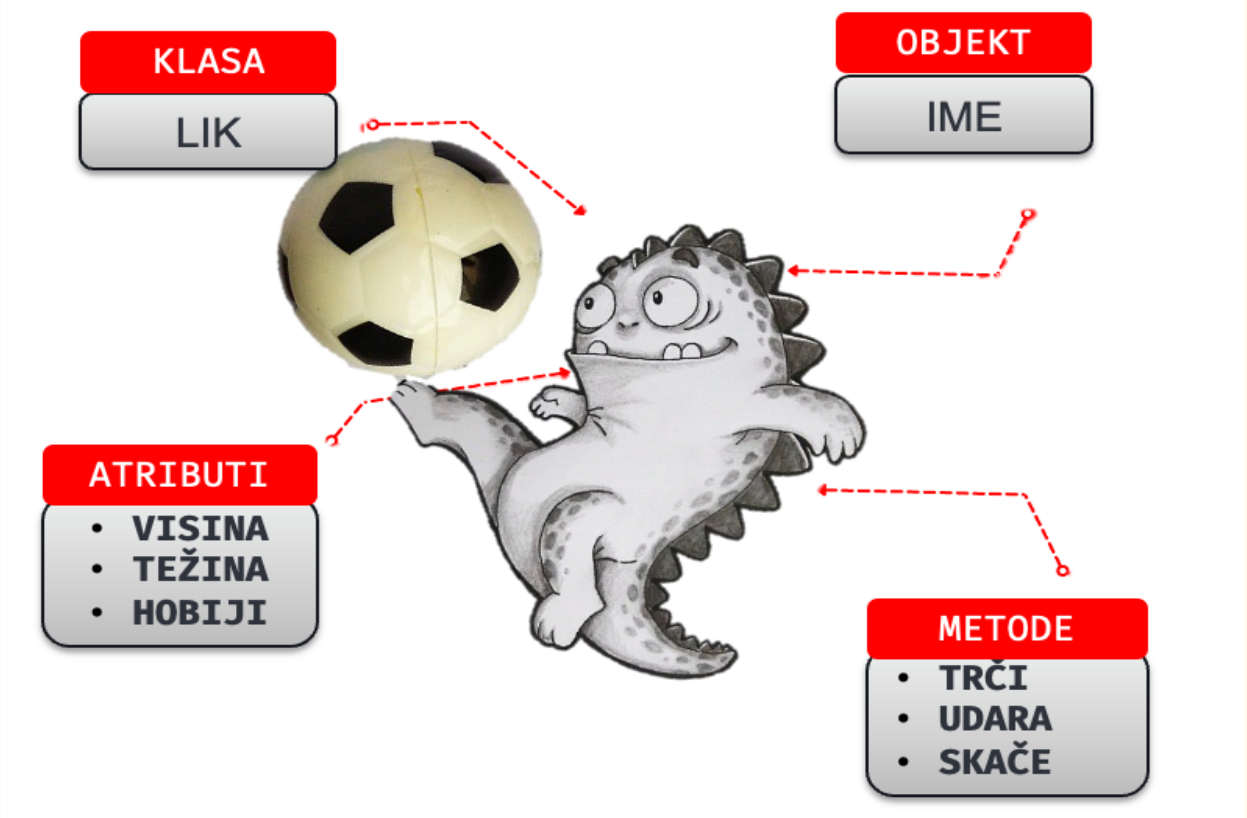
# UVOD
# OSNOVNI POJMOVI

**Obradili: Saša Mladenović, Divna Krpan i Goran Zaharija**



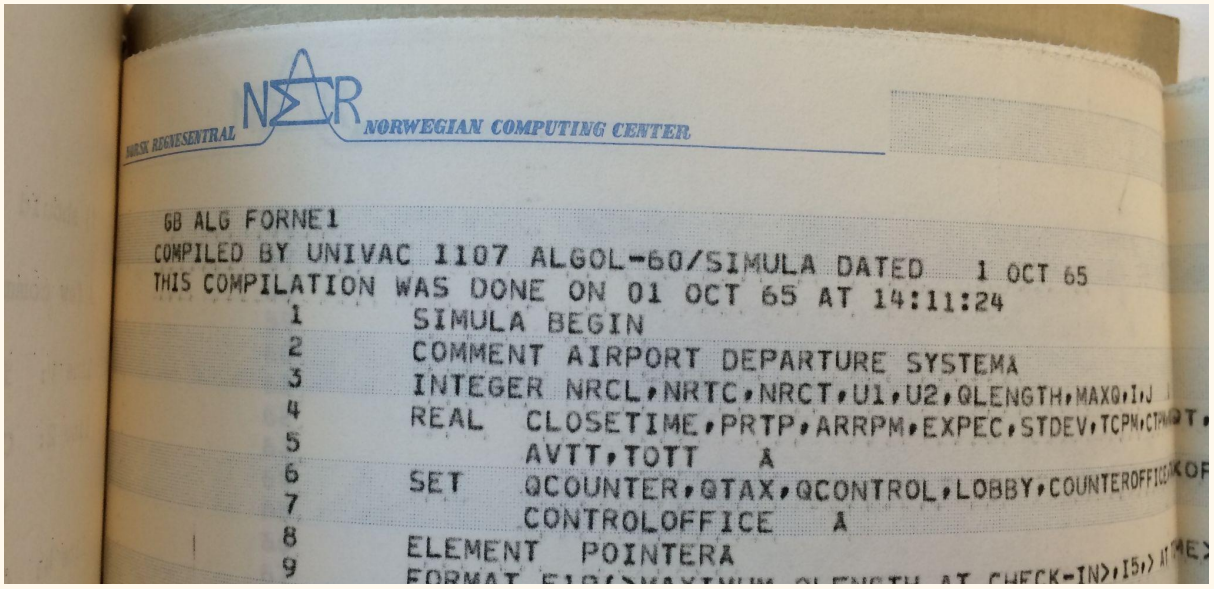

## UVOD

Mnogi vjeruju kako je OOP proizvod osamdesetih godina dvadesetog stoljeća i pomaka C-a prema objektno orijentiranom svijetu C++ programskog jezika kojeg je osmislio Bjarne Stroustrup (1980.). Ipak, programski jezici SIMULA 1 (1962.) i SIMULA 67 (1967.) najstariji su objektno orijentirani jezici. Ole-John Dahl i Kristen Nygaard sa Norveškog centra za računarstvo u Oslu izradili su programske jezike Simula.

Simula je akronim za *Simulation Language*, a programski jezik osmišljen je kako bi odredio način na koji programi mogu modelirati ili simulirati svijet na način na koji ga vide korisnici računala. Prema mišljenju eksperata, Simula je promijenila način izražavanja pomoću programskih jezika jer se usmjerila na objekte umjesto na linearne funkcije ili procedure.

Uporabom ovog programskog jezika objekti su mogli komunicirati putem poruka te su na taj način omogućili modelirati tradicionalnu ideju primatelja i pošiljatelja.

Nedvojbeno je kako je uporaba Simula programskog jezika dovela do uvođenja nekoliko drugih programskih jezika poput Jave, C++ i drugih novih objektno orijentiranih sustava i jezika.




Slika 1 - Prvi objektno orijentirani program

Iako je veći broj prednosti objektno orijentiranog programiranja bio dostupan već u Simula programskim jezicima, ovaj se način programiranja počeo značajnije koristiti u devedesetim godinama dvadesetog stoljeća, uvođenjem u širu primjenu programskog jezika C++.

Iz programskog jezika C nastao je C++, a često se znalo komentirati kako je C toliko moćan jezik da nema nikakav problem upucati se više puta u nogu. C++ je s druge strane toliko moćan da nije problem upucati se u nogu već ako to učinite ostat ćete bez nje vrlo jednostavno. Većina programera će priznati kako je C++ moćan programski jezik, ali jezik koji sa sobom donosi i cijeli niz složenosti. Programeri su tražili jezik koji je jednostavniji i možda manje složen, a koristi se za objektno orijentirano programiranje.

Objektno orijentirano programiranje dugo se smatralo naprednim programiranjem te se za njegovo poučavanje i usvajanje očekivalo dobro poznavanje proceduralnog programiranja. S obzirom na trend brzih promjena u programiranju, možemo reći kako je objektno orijentirana paradigma nužnost za svakog tko će se na neki način baviti programiranjem.

Često se ističe prednost prethodnog poznavanja programiranja u jednom ili više programskih jezika. Ipak, više nije svaki puta i bolje.

Pri savladavanju koncepata OOP-a oni koji su već upoznati s nekim načinom programiranja (paradigmom), npr. proceduralnom, imaju poteškoće s prijelazom na objektno-orijentiranu paradigmu jer sadrži koncepte koji se preklapaju, ima nove koncepte te zahtijeva drugačiji način razmišljanja i organizacije kôda. Najčešći je problem usvojiti nove nazive za identične ili gotovo identične programske konstrukte u drugom programskom jeziku.

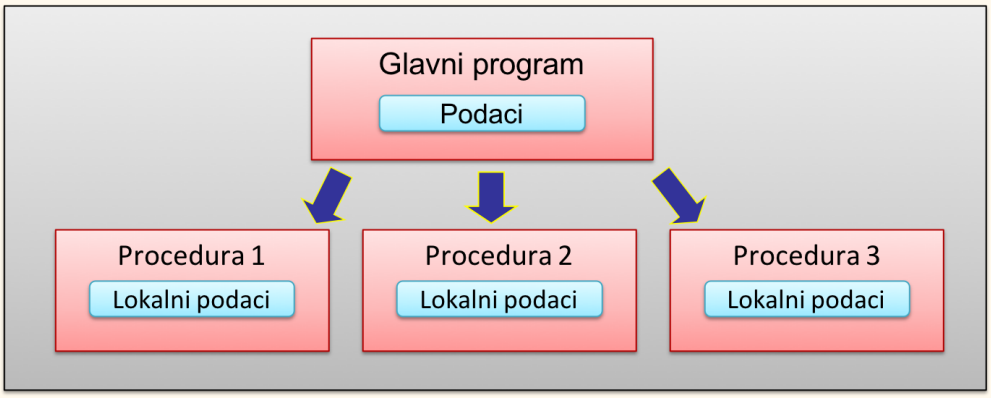
## Načini programiranja ili kako su se razvijali programski jezici

Početno programiranje, danas uobičajeno u Pythonu koristi za izradu prvih programa. Program je predstavljen nizom naredbi u glavnom dijelu programa. Programi obično obrađuju problem iz matematičke domene, poput izračuna faktorijela ili rješenja kvadratne jednadžbe i slično. Matematički kontekst odabran je iz povijesnih razloga, iz doba kada su računala služila za izračune, te zbog jednostavne povezanosti sa prethodnim znanjem, uglavnom iz matematike. Učenje programiranja, tako započinje pisanjem malih i jednostavnih programa koji se sastoje od slijeda naredbi i djeluju nad zajedničkim skupom podataka. Takvo programiranje nazivamo još i *Nestrukturirano programiranje*. Ponavljanje nekog posla znači ujedno i kopiranje naredbi, instrukcija koje posao opisuju. Ovakvo programiranje danas se često naziva Copy/Paste programiranje i unatoč tome što program možda i radi ispravno, smatra se nepoželjnim načinom programiranja.


Slika 2 - Nestrukturirano programiranje

Porastom zahtjeva, broj naredbi u programu raste, te se javlja potreba za nekom vrstom reorganizacije kôda, s ciljem njegova lakšeg praćenja. Naredbe koje čine neku logičku, smislenu, cjelinu grupiraju se, povezuju, u blokove kôda od kojih svaki čini proceduru ili funkcija. Naziv procedura, dolazi iz života gdje se od najranijeg školovanja uvode procedure, skup naredbi koje čine neku cjelinu. Prisjetite se procedure pranja zubi ili kuhanja kave. Kada su procedure napisane, u programu više nema potrebe za navođenjem svakog pojedinog koraka, već je moguće pozvati proceduru, nakon čega se podrazumijeva izvršavanje svih koraka navedenih u proceduri. Organiziranje kôda u procedure poboljšava čitljivost koda te smanjuje broj ponavljanja istih naredbi ili blokova naredbi.

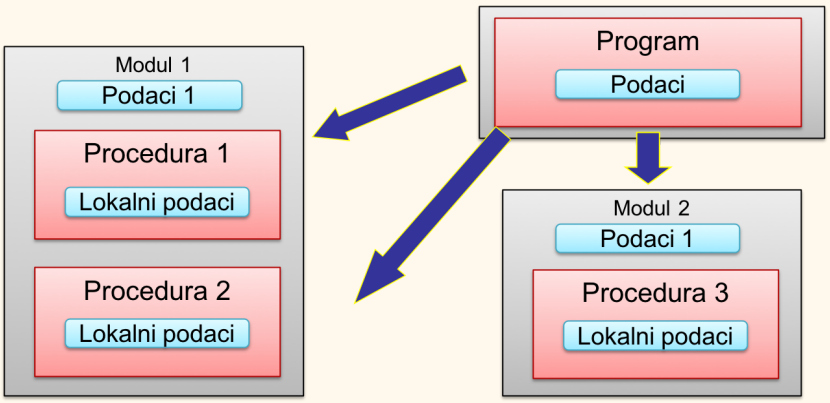
Izdvajanjem naredbi u procedure, program postaje slijed poziva procedura, a takav način programiranja naziva se Proceduralno programiranje. Svaka procedura upravlja lokalnim skupom podataka. Glavni program upravlja svojim skupom podataka.




Slika 3 - Proceduralno programiranje

Kada se za programiranje odabire proceduralna paradigma, glavni fokus je definirati procedure i odrediti kada pojedinu proceduru treba pozvati iz glavnog programa. Problem kod učenja proceduralnog programiranja nažalost svodi se često na problem usvajanja sintakse programskog jezika, a ne na promišljanje o načinu na koji će se problem razbiti na procedure ili funkcije koje se kasnije pozivaju u glavnom programu. Dodatno je problem objasniti iz kojeg razloga nije dobro ponavljati kôd u programu, ako je program duljine do stotinjak linija kôda.

Daljnjim porastom zahtjeva prema programskoj podršci, pojavljuje se potreba za dodatnim uvođenjem reda u programski kod. Osim urednosti javlja se i potreba za uvođenjem timskog rada pri programiranju. Navedeno dovodi do reorganizacije razvoja programske podrške koja se temelji na idejama proceduralne paradigme programiranja, a to je Modularno programiranje.

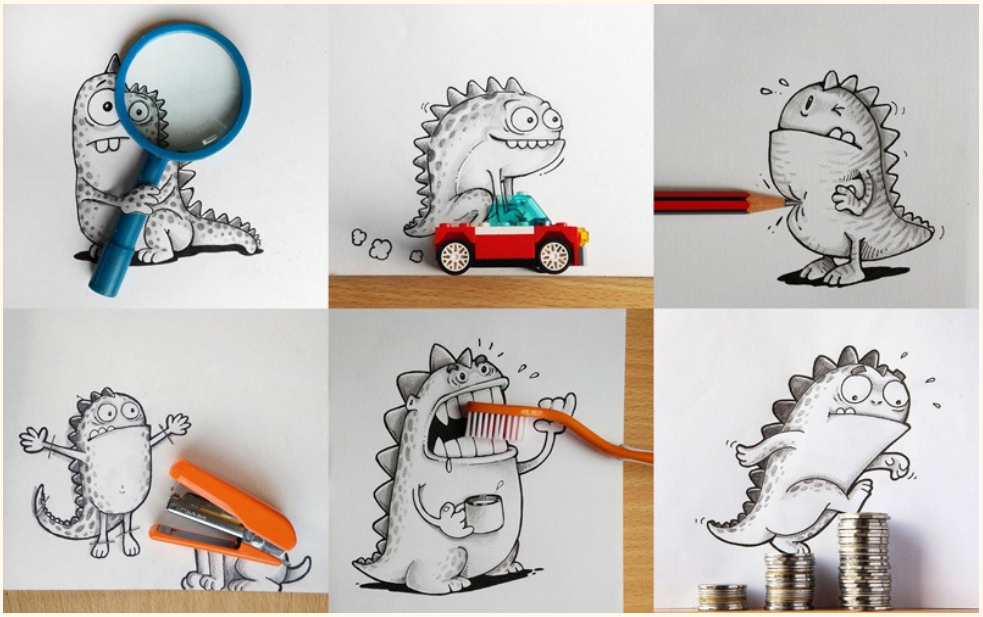



Slika 4 - Modularno programiranje

Kod modularnog programiranja procedure srodne funkcionalnosti grupiraju se u module koji mogu imati vlastite podatke. Glavni program poziva module od kojih je svaki zadužen za skupinu poslova.

Pojam objektno orijentirano programiranje skovao je Alan Curtis Kay, američki znanstvenik koji je u više navrata obrazložio kako je došao do ove ideje. Temeljnu ideju naslijedio je iz biologije i to iz činjenice da je svaka stanica samostalni entitet koji s ostalim stanicama ostvaruje interakciju putem "poruka", bez znanja o tome što ostale stanice zaista rade. Sve te stanice, autonomni entiteti zajedno tvore živi organizam. Uzimajući u obzir navedeno, smatrao je kako će ovakav način podjele odgovornosti, na veliki broj entiteta koji će brinuti o samima sebi i komunicirati samo slanjem poruka, pojednostaviti organizaciju programa.

Za razliku od proceduralnog programiranja, na koje ste navikli kod početnog programiranja, objektno orijentirano programiranje, (OOP), dodaje razinu logičke apstrakcije te umjesto procedura koristi koncept objekta kao osnovnog elementa za izgradnju programa. Objekt u OOP je logička reprezentacija nekog entiteta kojeg programer koristi u svom programu, a obično se sastoji od nekoliko različitih dijelova. Objektno orijentirani programi ne izvršavaju se slijedno, sekvencijalno, već ovise o događajima (eng. event-driven), a fokus je na interakciji među objektima, entitetima, sukladno onome što je izložio Key. Originalno, se objektno orijentirano programiranje koristilo kod izrade grafičkog korisničkog sučelja i kod izrade simulacija. Možemo reći kako je kod OOP-a naglasak stavljen na modeliranje, a tek potom na kodiranje.



Slika 5 - Objekti i modeliranje

Ako se pregleda literatura koja je nastala od dalekih šezdesetih godina prošlog stoljeća, pa sve do danas, moguće je izdvojiti ključne koncepte objektno orijentiranog programiranja: objekti, klase, nasljeđivanje, učahurivanje, metode, slanje poruka, polimorfizam i apstrakcija.

### NAPOMENA - Osnovni načini programiranja

- Nestrukturirano programiranje karakterizira slijed naredbi koje djeluju nad zajedničkim skupom podataka.
- Proceduralno programiranje program, koji upravlja svojim skupom podataka, pretvara u slijed poziva procedura od kojih svaka upravlja lokalnim skupom podataka.
- Modularno programiranje slijedi logiku razbijanja na procedure koje su više razine apstrakcije pa ih nazivamo moduli, ali se nakon toga moduli razbijaju na procedure. Program je postao slijed poziva modula.
- Objektno orijentirano programiranje umjesto procedura koristi koncept objekta i interakcije među njima kao osnovni gradivni element programa.

## Osnovni elementi objektno orijentiranog programiranja

Na temelju pregleda literature iz razdoblja od 1966-2005 u izdvojeni su ključni koncepti objektno orijentiranog programiranja kako slijedi:

- nasljeđivanje,
- objekti,
- klase,
- učahurivanje,
- metode,
- slanje poruka,
- polimorfizam i
- apstrakcija.

Temeljem navedenog može se definirati taksonomija osnovnih elemenata objektno orijentiranog programiranja koja se može prikazati tablicom

| Konstrukt | Koncept | Opis |
|---|---|---|
| Struktura | **Apstrakcija** | Izdvajanje klasa s ciljem pojednostavljivanja aspekata stvarnog svijeta |
| | **Klasa** | Opis organizacije i akcija zajedničkih jednom ili više sličnih objekata |
| | **Učahurivanje** | Oblikovanje klasa koje ograničavaju pristup podacima i ponašanju definiranjem ograničenog skupa poruka koje objekt može primiti |
| | **Nasljeđivanje** | Podaci i ponašanje jedne klase su uključeni u drugu klasu ili su osnova za drugu klasu |
| | **Objekt** | Individualni element kojeg je moguće identificirati, može biti stvaran ili apstraktan, sadrži svoje podatke i načine manipulacije tim podacima |
| Ponašanje | **Slanje poruka** | Objekt šalje podatke drugom objektu ili traži pokretanje metode drugog objekta |
| | **Metoda** | Način pristupa, postavljanja ili manipulacije informacijama sadržanima u objektu |
| | **Polimorfizam** | Različite klase mogu reagirati na istu poruku, a svaka od njih s drugačijom implementacijom |

Krenete li s pretragom knjiga, tutoriala i ostalih materijala koji se bave objektno orijentiranom paradigmom programiranja, često ćete se zbuniti jer su temeljni koncepti različito objašnjeni.

Koncepti se predstavljaju na različite načine koji ovise o kognitivnim zahtjevima prema ciljnoj populaciji, načinu modeliranja (npr. mape koncepata, UML), programskim jezicima (npr. Python, Java, C#), pseudokodu ili formalnom matematičkom prikazu. U nastavku evo nekoliko primjera objektno orijentiranih koncepata koji ovise o načinu implementacije pa ih u ovom trenutku nećemo definirati.

Objektno orijentirani programski jezik mora programeru omogućiti definiranje novih klasa te također implementaciju koncepata objektno orijentirane paradigme, ali to nije uvijek jednostavno. Način definiranja svakog koncepta prilično ovisi o odabranom programskom jeziku. Na primjer, konstruktor u programskom jeziku Python uvijek ima isti naziv: __init__, dok se u C#-u mora zvati jednako kao i klasa. Nadalje, ako promatramo način implementacije preopterećenih konstruktora u Javi i C#-u onda bi mogli reći da preopterećenje konstruktora u Pythonu nije podržano, no postoje tehnike kojima se to može napraviti. Problem je što ne postoji jedinstven način na koji se to radi te je zapravo jednostavnije reći da mora biti jedan konstruktor.

Zbog svega navedenog zaključujemo kako koncepte objektno orijentiranog programiranja moramo promatrati u odgovarajućem i konkretnom kontekstu pa ćemo u skladu s tim definirati promatrane koncepte objektno orijentiranog programiranja i njihovu reprezentaciju u programskim jezicima C# i JavaScript. Za koncepte programskog jezika koji ovdje nisu eksplicitno definirani vrijede pravila i definicije programskih jezika C# i JavaScript.

Svakako treba naglasiti kako nisu svi koncepti podržani na isti način, pa će se iz tog razloga prikazati primjeri realizacije u oba programska jezika te će se istaknuti razlike između njih. Ovi jezici nisu odabrani slučajno, već upravo zbog svojih različitosti u načinu interpretacije pojedinih koncepata objektno orijentiranog programiranja. Kako ste se već susreli s programskim jezikom C#, odlučili smo se detaljnije se pozabaviti JavaScript programskim jezikom i to dijelom koji se odnosi na objektno orijentiranu programsku paradigmu.