Uvod u programske paradigme

Jezik je skup pravila za komunikaciju izmedju subjekata. Pomocu jezika se predstavljaju I prenose informacije. Prirodni jezik se koristi za komunikaciju izmedju ljudi.
Postoji vise definicija programskog jezika:
→ Programski jezik je skup sintaktickih I semantickih pravila koja se koriste za opis (definiciju) racunarskih programa.
→ Programski jezik je jezik konstruisan formalno da bise omogucilo zadavanje intstrukcija masinama, posebno racunarima.
→ Programski jezik je vestacki jezik koji sluzi za opis racunarskih programa.
→ Programski jezik je vestacki jezik za opis konstrukcija koje mogu biti prevedene u masinski jezik I izvrsene od strane racunara.

Programski jezici se koriste za komunikaciju ljudi I racunara, medjusobnu komunikaciju medju racunarima ali I ljudima. Postoji veliki broj programskih jezika, broj se krece u hiljadama (preko 2000), medjutim nisu svi jezici podjednako vazni I zastupljeni.
TIOBE index predstavlja indikator popularnosti programskih jezika. Indeks se racuna jednom mesecno na osnovu velikog broja faktora koji daju celokupnu sliku o popularnosti jezika, kao sto su broj korisnika, broj odrzanih kurseva, zastupljenost na Gitu I slicno …

Rec paradigma oznacava uzor, obrazac, sablon I obicno se koristi da oznaci vrstu objekata koji imaju zajednicke karakteristike. Programska paradigma predstavlja odredjeni programski stil, sablon, odnosno nacin programiranja. Broj programskih paradigmi nije belki kao broj programskih jezika I izucavanjem odredjene paradigme upoznajemo globalna svojstva jezika koji pripadaju toj paradigmi. Samim tim poznavanje paradigme znacajno olaksava ucenje programskog jezika koji joj pripada. Svakoj programskoj paradigmi pripada vise programskih jezika I potrebno je izuciti svojstva najistaknutijih predstavnika programskih paradigmni (“Koliko predstavnika razlicitih paradigmni znas, toliko vredis”). Takodje, jedan programski jezik moze podrzati vise razlicitih paradigmi.
Na primer, C++ podrzava proceduralni, objektno-orijentisani, funkcionalni I jos neke stilove programiranja.

Prvi elektronski računari nastali su krajem 40-ih godina 20. veka, kao što je ABC koji se koristio za rešavanje sistema linearnih jednačina.
Krajem 1940. godine dolazi do konceptualne promene u vidu fon Nojmanove arhitekture i do razdvajanja hardvera i softvera. Programske jezike u skladu sa tim možemo podeliti na mašinski zavisne, na primer asembleri, i na mašinski nezavisne, na primer Python, C++ itd. Prednosti mašinski zavisnih programskih jezika su veća kontrola nad memorijom i načinom izvršavanja konkretnih operacija, a mane to što su komplikovaniji, teži za razumevanje, zahtevaju mnogo vremena za kodiranje osnovnih operacija i nisu prenosivi, odnosno zavise od konkretne mašine za koju je program pisan. Prednosti mašinski nezavisnih jezika su lakše razumevanje, činjenica da su sličniji prirodnim jezicima i da prilikom kodiranja ne moramo razmišljati o konkretnom načinu izvršavanja osnovnih operacija, prenosivost i brže kodiranje, a mane to što sa druge strane gubimo kontrolu nad memorijom i izvršavanjem konkretnih operacija.
Prvi programski jezici nastaju krajem 50-ih godina 20. veka i to su bili:
→ FORTRAN koji se koristio za matematička izračunavanja,
→ LISP koji je bio zasnovan na funkcionalnoj paradigmi i kome je najduže vremena trebalo da se razvije I bude prihvaćen
→COBOL koji je napravljen za ljude koji nisu bili matematičari i programeri i koristio se u

preduzećima, fabrikama i slično. U istom periodu nastaje i ALGOL koji se koristio za različita izračunavanja. Šezdesetih godina nastaju Simula i Basic, 70-tih C, Pascal i Prolog, 80-tih C++ i Erlang, 90-tih Haskell, Python, PHP, Ruby, JavaScript itd. Tokom 2000-tih nastaju C#, F#, Scala, Elixir i mnogi drugi. U početku je svaki jezik pripadao samo jednoj paradigmi, a kasnije su jezici implementirali funkcionalnosti različitih paradigmi. Nastanak i razvoj programskih jezika se može prikazati pomoću razvojnog stabla, gde je prikazano kako i kada se koji jezik razvijao i pod uticajem kojih drugih jezika.

Nove programske paradigme nastajale su uz teznju da se olaksa proces programiranja. Istovremeno, nastanak novih paradigmi vezan je sa efikasnim kreiranjem sve kompleksnijeg softvera.
Svaka novonastala paradigma bila je promovisana preko nekog programskog jezika. Razvoj programskih I paradigmi I jezika povezan je I sa razvojem hardvera.

Ne postoji jedinstveno shvatanje programskih paradigmi, samim tim ni jedinstvena podela. Najopstija podela programskih paradigmi je na proceduralnu gde je osnovni zadatak programera da opise nacin kojim se dolazi do resenja problema I deklarativnu paradigmu, gde je osnovni zadatak programera da precizno opise problem, dok se mehanizam programskog jezika bavi pronalazenjem resenja problema.

U osnovne programske paradigme ubrajamo:
→ imperativna paradigma
→ objektno-orijentisana paradigma
→ funkcionalna paradigma
→ logicka paradigma

Ostale paradigme se cesto tretiraju kao podparadigme ili kombinacije osnovnih. Neke su:
→ komponentna paradigma
→ paradigma upitnih jezika
→ paradigma programiranja ogranicenja
→ genericka paradigma
→ vizuelna paradigma
→ konkurentna paradigma
→ skript paradigma

Napomena o imperativnoj I proceduralnoj paradigmi
Postoji vise shvatanja proceduralne paradigme:

1) Proceduralna paradigma je svaka paradigma kod koje se u procesu programiranja opisuje algoritam tj procedura resavanja problema. U ovom slucaju je imperativna paradigma podparadigma proceduralne paradigme

2) Proceduralna paradigma je podparadigma imperativne paradigme koju karakterise, pored naredbi, I njihovo grupisanje u podprograme(funkcije). U ovom slucaju, u literaturi se cesto imperativna I proceduralna paradigma koriste kao sinonimi.

--/----------

Programski jezik je sredstvo koje koristi ***covek*** da izrazi ***proces*** pomocu kojeg ***racunar*** resava nekakav ***problem***. U zavisnosti od toga na kojoj od ovih reci je akcenat, programskim jezikom je podrzana dominantna programska paradigma:

→ ***covek***: logicka paradigma
→ ***proces***: funkcionalna paradigma
→ ***racunar***: imperativna paradigma
→ **problem**: objektno-orijentisana paradigma

--/----------

Osnovne programske paradigme

Imperativna paradigma

Imperativna paradigma nastala je pod uticajem Fon Nojmanove arhitekture racunara. Moze se reci da se zasniva na tehnoloskom konceptu digitalnog racunara. Proces izracunavanja se odvija slicno kao neke svakodnevne rutine (zasnovan je na algoritamskom nacinu rada), kao sto je spremanje hrane koriscenjem recepata itd..
Moze se okarakterisati recenicom “prvo uradi ovo, zatim uradi ono”
Procedurom se saopstava racunaru KAKO se problem resava, tj navodi se precizan niz koraka (algoritam) potreban za resavanje problema.
Osnovni pojam imperativnih jezika je naredba.
Naredbe se grupisu u procedure I izvrsavaju se sekvencijalno ukoliko se eksplicitno u programu ne promeni redosled izvrsavanja.
Upravljacke strukture su naredbe grananja, iteracije, skoka (goto).
Oznake promenljivih su oznake memorijskih lokacija pa se u naredbama cesto mesaju oznake lokacija I vrednosti – to izaziva bocne efekte.
Jezici: C, Pascal, Basic, Fortran …

Objektno-orijentisana paradigma

Ovo je jedna od najpopularnijih programskih paradigmi. Simulira (modeluje) spoljasnji svet pomocu objekata. Objekti interaguju medjusobno.
Mogla bi da se okarakterise recenicom: “Uputi poruku objektima da bi simulirao tok nekog dogadjaja”
Podaci I procedure se enkapsuliraju u objekte.
Koristi se skrivanje podataka da bi se zastitila unutrasnja svojstva objekta.
Objekti su grupisani po klasama koje predstavljaju sablon/koncept po kojem se kreiraju objekti.
Klase najcesce hijerarhijski organizovane I povezane mehanizmom nasledjivanja.
Jezici: C++, Java, C# …

Funkcionalna paradigma

Rezultat teznje da se drugacije organizuje proces programiranja. Izracunavanja su evaluacije matematickih funkcija. Zasnovana je na pojmu matematicke funkcije I ima formalnu strogo definisanu matematicku osnovu u lambda racunu.
Moze se okarakterisati recenicom: “Izracunati vrednost izraza I koristiti je”
Eliminisani su bocni efekti sto utice na lakse razumevanje I predvidjanje ponasanja programa – izlazna vrednost funkcije zavisi samo od ulaznih vrednosti argumenata.
Predstavnici: Lisp I Haskell

Logicka paradigma

Nastaje kao teznja da se u kreiranju programa koristi isti nacin razmisljanja kao i pri resavanju problema u svakodnevnom zivotu
→ Deklarativna paradigma
Opisuju se odnosi izmedu cinjenica i pravila u domenu problema; koriste se aksiome, pravila izvodjenja i upiti.
Logicka paradigma se dosta razlikuje od svih ostalih po nacinu pristupa resavanju problema.
Nije jednako pogodna za sve oblasti izracunavnja, osnovni domen je resavanje problema vestacke inteligencije.
Izvrsavanje programa se zasniva na sistematskom pretrazivanju skupa cinjenica uz koriscenje odredjenih pravila zakljucivanja.
Zasnovana na matematickoj logici, tj. na predikatskom racunu prvog reda.
Zasnovana je I na automatskom dokazivanju teorema (metod rezolucije).

Moze se okarakterisati recenicom: “Odgovori na pitanje kroz trazenje resenja”
Predstavnik: Prolog, Asp, Datalog …

Dodatne programske paradigme

Skript Paradigma

Skript jezik je programski jezik koji sluzi za pisanje skriptova. Skript je spisak (lista) komandi koje mogu biti izvrsene u zadatom okruzenju bez interakcije sa korisnikom.
Skript jezici mogu imati specifican domen primene:

- → U prvobitnom obliku pojavljuju se kao komandni jezici operativnih sistema (npr Bash)
- → Velika primena u programiranju web aplikacija (JavaScript, TypeScript …)
- → Cesto se koriste I za povezivanje komponenti unutar neke aplikacije

Skript jezici mogu biti I jezici opste namene (npr Python).
Nije uvek lako napraviti razliku izmedju skript-jezika I drugih programskih jezika.
Skript paradigma je cesto specificna kombinacija drugih paradigmi, kao sto su: objektno-orijentisana, proceduralna, funkcionalna pa je to razlog sto se skript paradigma ne prepoznaje uvek kao posebna paradigma.
Skript jezici imaju razne specificne osobine:
→ Obicno se ne kompiliraju vec interpretiraju
→ Obicno dinamicki odredjuju tipove
Skript jezici su u ekspanziji: Unix Shell (sh), JavaScript, PHP, Python …

Paradigma programirananja ogranicenja

U okviru paradigme programiranja ogranicenja zadaju se relacije izmedju promenljivih u formi nekakvih ogranicenja. Ogranicenja mogu biti raznih vrsta, npr. logicka ili linearna.

Deklarativna paradigma: ova ogranicenja ne zadaju sekvencu koraka koji treba da se izvrse vec osobine resenja koje treba da se pronadje.

Jezici za programiranje ogranicenja cesto su nadogradnja jezika logicke paradigme, na primer Prologa.
Postoje biblioteke za podrsku ovoj vrsti programiranja u okviru imperativnih/objektno-orijentisanih/skript programskih jezika (C, Java, C++, Python …)
Programiranje ogranicenja takodje moze biti I sastavni deo jezika (Bprolog, Curry)

Komponentna paradigma

Ideja je da se softver sklapa od vecih gotovih komponenti, kao sto se to radi kod sklapanja elektronskih I tehnickih uredjaja. Komponenta je jedinica funkcionalnosti sa “ugovorenim” interfejsom.
Interfejs definise nacin na koji se komunicira sa komponentom, I on je u potpunosti odvojen od implementacije.
Komponentna paradigma predstavlja priblizavanje deklarativnom stilu programiranja.
Cilj je da se uprosti proces programiranja I da se jednom kreirane komponente mnogo puta koriste.

Komponentna paradigma se može posmatrati kao potparadigma objektno-orijentisane paradigme. Obe paradigme dele nekoliko ključnih koncepata, kao što su enkapsulacija, ponovna upotreba koda i modularnost. Međutim, postoji nekoliko specifičnih aspekata koji komponentnu paradigmu izdvajaju, jedan od njih je ***enkapsulacija na visem nivou***.
Encapsulacija na višem nivou: Dok objektno-orijentisana paradigma fokusira na enkapsulaciju stanja i ponašanja unutar objekata, komponentna paradigma ide korak dalje tako što inkapsulira više klasa i drugih resursa (kao što su biblioteke ili servisi) u komponente koje nude jasno definisane interfejse.

Softverska komponenta je kolekcija delova (metoda I objekata) koji obezbedjuju neku funkcionalnost. Kao I tehnicke komponente, softverske komponente mogu biti proste ili kompleksne, delovati samostalno ili u konjukciji sa drugim jedinicama.

Komponente se medjusobno povezuju da bi se kreirao kompleksan softver pritom nacin povezivanja komponenti treba da bude jednostavan. Najjednostavniji pristup je prevlacenjem i spustanjem na zeljenu lokaciju, tj kada se kreiranje programa vrsi biranjem komponenti i postavljanjem na pravo mesto, a ne pisanjem linije za linijom. Na primer: kreiranje interfejsa aplikacija (postavljanje prozora, tekstualnih polja, dugmica …)

U okviru komponentnog programiranja, vazno je razvojno okruzenje koje se koristi, dok sama implementacija komponenti i kod koji se komponentnim programiranjem generise moze da bude u razlicitim programskim jezicima, npr JAVA, C++, C# …

Paradigma upitnih jezika

Upitni jezici mogu biti vezani za baze podataka ili za pronalazenje informacija. Paradigma upitnih jezika je deklarativna paradigma.
Upitni jezici baze podataka: na osnovu struktuiranih cinjenica zadatih u okviru struktuiranih baza podataka daju konkretne odgovore koji zadovoljavaju nekakve trazene uslove.
Najpoznatiji predstavnik upitnih jezika je SQL koji se koristi za relacione baze podataka (select, from, where).

Primer) Iz tabele studenti izvuci sva imena studenata koji se prezivaju Petrovic:
```
SELECT ime
FROM studenti
WHERE prezime=”Petrovic”;
```

SPARQL je jezik koji se koristi za podatke u RDF (Resource Description Framework) formatu (select, from, where).
Primer) Izdvoj sve studente osnovnih studija, kurseve koji oni slusaju i njihova imena
```
select ?x ?y ?n
where {
        ?x a :UndergradStudent .
        ?x :takesCourse ?y .
        ?x :name ?n
}
```

XQuery je jezik za pretrazivanje XML struktuiranih podataka (for, let, where, order by, return)
Primer) Za svaku knjigu iz prodavnice za koju se informacije cuvaju u okviru dokumenta books.xml, i za koju vazi da je skuplja od 30, vrati njen naslov u sortiranom redosledu po naslovima
```
for $x in doc("books.xml")/bookstore/book
where $x/price>30
order by $x/title
return $x/title
```

Genericka paradigma

Genericko programiranje je stil programiranja gde se algoritmi pisu sa apstrahovanim tipovima. Tipovi se obezbedjuju kao parametri I time se izbegava dupliranje koda. Na primer, algoritmi trazenja minimuma, maksimuma, sortiranja, pretrage …
Ideja nastala u jeziku ML.
Generics u jezicima Python, C#, F#, Java …
Sabloni (templates) u jezicima C++ i D.
Implicitni polimorfizam u jezicima ML, Scala …

Vizuelna paradigma

Vrsi modelovanje spoljasnjeg sveta, usko povezana sa objektno-orijentisanom paradigmom. Koriste se graficki elementi (dijagrami) za opis akcija, svojstva I povezanosti sa raznim resursima.
Vizuelni jezici dominantni u fazi dizajniranja programa, pogodnija za pravljenje “skica” programa, ne za detaljan opis.
Glavni predstavnik ove paradigme je UML (engl. Unifed Modeling Language).

Postoje softverski alati za prevodjenje “vizuelnog opisa” u neki programski jezik (samim tim I u izvrsivi kod).

• Primeri strukturnih informacija:
dijagram klasa,
dijagram objekata,
dijagram komponenti.

• Primeri informacija o ponasanju:
dijagram komunikacije
dijagram aktivnosti
dijagram slucajeva upotrebe

Konkurentna paradigma

Konkurentnu paradigmu karakterise vise procesa koji se izvrsavaju u istom vremenskom periodu, a koji imaju isti cilj. Postoje razlicite forme konkurentnosti:
→ Konkurentnost u uzem smislu: jedan procesor, jedna memorija
→ Paralelno programiranje: vise procesora, jedna memorija
→ Distribuirano programiranje: vise procesora, vise memorija

Konkurentnost u uzem smislu karakterise preklapajuce izvrsavanje vise procesa koji koriste isti procesor i koji komuniciraju preko zajednicke memorije. Ovi procesi modeluju procese spoljasnjeg sveta koji mogu da se dese konkurentno, na primer kod operativnih sistema. Ukoliko postoji vise procesora sa pristupom jedinstvenoj memoriji, onda je u pitanju paralelno programiranje. Ukoliko postoji vise procesora od kojih svaki ima svoju memoriju, onda je u pitanju distribuirano programiranje - procesi medjusobno salju poruke da bi razmenili informacije. Moze se shvatiti kao vrsta paralelnog izracunavanja ali sa drugacijom medjusobnom komunikacijom koja namece nove izazove.

Pisanje konkurentnih programa je znacajno teze od pisanje sekvencijalnih. Namecu se novi problemi, po pitanju sinhronizacije procesa I pristupa zajednickim podacima.
Prisutno je u skoro svim savremenim programskim jezicima: Java, C, C++ …

Reaktivna paradigma

Rekativno programiranje je usmereno na tok podatka u smislu prenosenja izmena prilikom asinhrone promene podataka. Na primer, u imperativnom programskom jeziku, a = b + c je komanda koja se izvrsava dodelom vrednosti promenljivoj a na osnovu trenutnih vrednosti promenljivih b i c i kasnija promena vrednosti b ili c ne utice na promenu vrednosti promenljive a.
Kod reaktivnog programiranja, a = b + c ima znacenje da svaka promena vrednosti b i c utice na izmenu vrednosti promenljive a (kao sto je to u okviru tabela, npr VisiCalc, Excel, LibreOffice Calc)

Poznavanje reaktivne paradigme je osnovno za pisanje veb servisa, mobilnih aplikacija, sistema sa real-time komponentama.
Osnovne karakteristike:
→ Reaguje na dogadjaje: event-driven (vodjena dogadjajima)
→ Reaguje na rast upotrebe: skalabilna (scalable)
→ Reaguje na padove: mogucnost brzog oporavka od pada (resilient)
    → Pad moze biti softverski izuzetak, hardverski pad ili pad veze.
    → Da bi se to ostvarilo, mora da bude sastavni deo arhitekture sistema (medjusobno slabo zavisne komponente)

Reaktivno programiranje nije vezano za programski jezik, vec je to stil programiranja koji se moze ostvariti u najrazlicitijim programskim jezicima !

Jezici za obeležavanje teksta/podataka

Postoje jezici za obeležavanje i strukturiranje teksta ili podataka, kao što su HTML, CSS, XML, SGML, JSON i drugi. Jezici za obeležavanje teksta i podataka **nisu programski jezici** jer se ne uklapaju u definiciju programskih jezika. Naime, po definiciji, programskim jezikom se zadaje neko izračunavanje, odnosno definišu se programi koje računar može da izvrši. Ovim jezicima definiše se struktura teksta ili podataka, ne pišu se programi i nema nikakvog izvršavanja. U skladu sa time, ovi jezici nisu programski jezici pa samim tim ne mogu da formiraju programsku paradigmu.

Na primer, HTML definiše strukturu veb stranice i određuje naslove, paragrafe i slično. Brauzer na osnovu tih informacija (i eventualno dodatnih stilskih informacija kroz CSS) određuje kako će podaci biti prikazani. Dakle, HTML se koristi za određivanje strukturnih osobina veb stranice, a ne njene funkcionalnosti. Programski jezici imaju funkcionalnu namenu. HTML, kao jezik za obeležavanje, ne može da uradi ništa u smislu u kojem programski jezici to rade jer ne sadrži logiku programiranja: ne sadrži kontrolu toka, evaluaciju izraza, funkcije, promenljive, pojam događaja i slično. Ne može da modifikuje podatke niti da ima ulaz/izlaz.

Paralelno sa razvojem jezika za obeležavanje (posebno XML), razvijeni su specijalizovani programski jezici za razne obrade koje se odnose na jezike za obeležavanje. U takve jezike spadaju: XSLT, XQuery, XLS i drugi. Ovi jezici se mogu pridružiti raznim paradigmama. Na primer, to su najčešće domenski specifični jezici i skript jezici.