|  |
| --- |
|  |
| **NASLOV DOMAĆEG ZADATKA \*** |
| Uvod u skripting jezike |

|  |  |
| --- | --- |
| **PROLOG** | |
| **Autor\*** | *Milan Hakaj* |
| **Kategorizacija\*** | IT2008-IPT-SCRP-STRL-V-V01 |
| **Nizo težine\*** | Osnovni |
| **Ko radi ?\*** | OAS student |

|  |
| --- |
| **SADRŽAJ** |
| **Naslov sekcije** |
| ***Uvod*** |
| **Sadržaj (telo) sekcije** |
| Skripting jezici imaju formu programskog jezika koji se uglavnom interpretira a ne kompajlira. Konvencionalni programi se pretvaraju u izvršne datoteke pre pokretanja. Nasuprot tome, programi napisani u skripting jezicima se interpretiraju i tumače liniju po liniju odnosno komandu po komandu u jednom trenutku. Skripting jezici su danas znatno napredniji od svojih daljih predaka, po pitanju performansi, sintakse i mogućnosti samih jezika. Ovi jezici su često pisani da olakšaju naprednije funkcije veb sajtova. Ove karakteristike se obrađuju na serveru ali se skripta na konkretnoj stranici izvršava u korisničkom veb čitaču. Tako skripting jezike možemo podeliti na one koji se izvršavaju na strani klijenta i one koji se izvršavaju na strani servera.  U većini slučajeva je lakše pisati kod u nekom od skripting jezika. Skripting jezici omogućavaju brz razvoj i lako komuniciranje sa programima pisanim u drugim programskim jezicima ili za pisanje skripti u shell okruženjima što je originalno bio I cilj.  Skripting jezici su evoluirali u ozbiljne jezike opšte namene. Samim tim, veliki broj jezika koje danas nazivamo skripting jezicima su u isto vreme i opšti programski jezici – postoje čitavi informacioni sistemi pisani u Python-u i Ruby-u, a u poslednjih par godina i u JavaScript-u (node.js serverske aplikacije). S druge strane, postoje i jezici koji se tipično ne smatraju skripting jezicima, ali se koriste kao takvi u sklopu drugih sistema. Tako, na primer, Unity3D koristi C# kao skripting jezik, iako je C# kompajliran, statički tipiziran jezik.  Skripting jezici se takođe mogu koristiti za kreiranje grafičkih korisničkih interfejsa i formi koji poboljšavaju praktičnost pretraživanja, veb-baziranih emailova i e-commerce, itd.  Neke od karakteristika jezika koji se često koriste kao skripting jezici su:  ●Dinamička tipiziranost – tipovi vrednosti su poznati tokom izvršenja, a statička provera tipova nije moguća ili je vrlo ograničena. Sskoro svi skripting jezici imaju ovu osobinu.  ●Laka integracija i proširenje unutar sistema pisanih u drugim jezicima. Lua i Python su posebno pogodni za integraciju i proširenja.  ●Veliki broj biblioteka za rad sa fajlovima, direktorijumima, drugim sistemskim komponentama. Python, Perl i Ruby imaju standardne biblioteke koje podržavaju mnoštvo funkcija za rad sa sistemskim resursima.  ●Mali memorijski footprint i brz i lagan runtime, posebno bitno za jezike čija je glavna namena ugrađivanje u druge programe. Lua je jedan od jezika čiji je memorijski footprint jako mali (manji od većine drugih skripting jezika) |
| **Naslov sekcije** |
| ***Različite vrste programiranja (Vrste programskih jezika po tipu)*** |
| **Sadržaj (telo) sekcije** |
| **Object-oriented programming**  Object-oriented programming je metod programiranja baziran na hierarhiji klasa I dobro definisanih objekata. Object-oriented programming je programska paradigma koja je organizovana oko objekata a ne akcija I podataka a ne logike.  **Imperative programming**  imperativno programiranje je programska paradigma koja opisuje izvršavanje nekog programa u pogledu sekvencijalnih izjava koja menjaju stanje programa. Imperativno programiranje dakle govori mašini kako da nešto uradi a kao rezultat desiće se ono što smo želeli. Dakle, kod imperativnog programiranja mi govorimo računaru "saberi x i y", i računar to radi. Ovo je jedan od najčešćih načina programiranja koji se pojavljuje gotovo u svim jezicima.   |  | | --- | | 1 var numbers = [1,2,3,4,5]  2 var doubled = []  3  4 **for**(var i = 0; i < numbers.length; i++) {  5 var newNumber = numbers[i] \* 2  6 doubled.push(newNumber)  7 }  8 console.log(doubled) *//=> [2,4,6,8,10]* |   Dakle, u primeru se eksplicitno vrši iteracija kroz svaki element u nizu, izdvajanje svakog pojedinačno i vrši udvostručavanje njegove vrednosti - potom se ova vrednost smešta u novi niz mutirajući udvostručeni niz korak po korak dok ne budemo gotovi.  **Deklarativno programiranje**  Deklarativno programiranje je programska paradigma koja predstavlja stil kreiranja strukture programa čija se logika treba ostvariti bez preciziranja kontrolnog toka odnosno načina na koji je to potrebno uraditi. Ukoliko se vratimo na primer imperativnog programiranja – više deklarativan način za kreirajne novog niza sa udvostručenim vrednostima bio bi korišćenjem Array.map funkcije:   |  | | --- | | 1 var numbers = [1,2,3,4,5]  2  3 var doubled = numbers.map(function(n) {  4 **return** n \* 2  5 })  6 console.log(doubled) *//=> [2,4,6,8,10]* |   **Procedural programming**  Proceduralno programiranje, koje se može nazvati imperativnim s obzirom da se korišćenjem ove programske paradigme takođe definiše set koraka koje program mora proći da bi došao do željenog stanja, poseduje uredni tok kontrole kroz program počevši od vrha pa do poslednje linije koda. Proceduralno programiranje korisno je kada je potrebno izvršiti određeni zadatak (uglavnom onaj koji se može koristiti višestruko). Dakle, definiše se kontrola u samoj funkciji koja izvršava specifični set akcija a potom vraća određenu vrednost.   |  | | --- | | 1 int add(int a, int b);  2 int main(){  3 int firstNum = 6;  4 int secondNum = 15;  5 int sum;  6 sum = add(firstNum,secondNum);  7 printf("sum= ",sum);  8 **return** 0;  9 }  10 int add(int a,int b){  11 int result;  12 result = a + b;  13 **return** result;  14 } |   Primer ovakvih jezika uključuju BASIC, Pascal, C, itd.  **Functional programming**  Funkcionalno programiranje fokusira se na primeni funkcija za razliku od imperativnog programiranja koji naglašava promene stanja. Bazira se na korišćenju matematičkih funkcija i izbegava stanja i promenljive podatke. Dakle, funkcionalno programiranje predstavlja programsku paradigmu kada se koriste funkcije, a ne objekti ili procedure, koje čine osnovne blokove programa. Funkcionalno programiranje nije vezano za matematiku već abstrakciju i smanjivanje kompleksnosti - čineći moćnu paradigmu koja se može pozabaviti kompleksnim, real-world programskim zadacima.  Jedan od dobrih primera koji može omogućiti dobar uvid u to šta predstavlja funkcionalno programiranje je QuickSort algoritam koji koristi PatternMatching, ListComprehensions i rekurziju u Haskell-u:   |  | | --- | | qsort [] = []  qsort (x:xs) = qsort elts\_lt\_x ++ [x] ++ qsort elts\_greq\_x  where  elts\_lt\_x = [y | y <- xs, y < x]  elts\_greq\_x = [y | y <- xs, y >= x] |   **Dynamic programming**  Ideja koja se krije iza ovog koncepta je krajnje jednostavna. U suštini, kako bi se rešio neki konkretan problem najpre se moraju rešiti različiti delovi tog problema (subproblems) a potom, kombinovanjem rešenja ovih različitih delova dolazi se do cilja odnosno do rešenja celokupnog problema. Uglavnom, kada se koriste druge metode, mnogi od potproblema su generisani I rešavani mnogo puta. Koncept dinamičkog programiranja ima za cilj rešavanje ovih problema težeći ka tome da reši svaki potproblem samo jednom I time smanji broj izračunavanja. Ovo znači da kada je jednom pronađeno rešenje za konkretan problem on se skladišti tako da kada je sledeći put potrebno rešiti isti problem potrebno je samo izvršiti pretraživanje. Ovo je svakako jako efikasno I korisno kada broj potproblema koji se javljaju raste eksponencijalno.  **Logic programming**  Bazira se na ideji korišćenja logičkih izraza za predstavljanje programa I za izvršavanje računarskih operacija. Način logičkog izraza koji se može naći u logičkom programiranju je definisan u obliku sledećeg izraza:  G if G1 and … and Gn |
| **Naslov sekcije** |
| ***Sistemi tipiziranja*** |
| **Sadržaj (telo) sekcije** |
| **Type-safety**  Type-safety je karakteristika jezika koji su tako dizajnirani da se podudaraju sa sada već dobro poznatim sloganom koji glasi "well-typed programms cannot go wrong". Ovaj slogan zapravo govori o tome da programi ne mogu pasti zbog runtime type error greške. Zapravo, to znači da parametri koji se primenjuju ne smeju imati vrednost nekompatibilnog tipa. To znači da će biti izvršena validacija tipova prilikom kompajliranja i prikazivanje greške ukoliko pokušamo da dodelimo pogrešan tip određenoj promenljivoj.   |  | | --- | | 1 *// Variable example: this will fail*  2 String text = 1;  3 int num = "abc"; |  |  | | --- | | 1 *// Method example: this will fail*  2 int add(int a, int b) {  3 **return** a + b;  4 }  5  6 int sum = add(5, "3"); |   Neki od type-safe programskih jezika su Java, Python, Ruby dok na primer C i C++ ne spadaju u ovu grupu. Međutim, potrebno je razumeti da “type-safety” ima specifično značenje zavisno od tipa sistema jezika o kome se govori.  **Dynamic/Static typing**  Statički tipizirani programski jezici su oni u kojima je promenljivu potrebno definisati pre nego što ona bude korišćena. Ovo podrazumeva da statičko tipiziranje ima veze sa eksplicitnim deklarisanjem ili inicijalizovanjem promenljivih pre nego što se one počnu koristiti u programu. Java je klasični predstavnik statički tipiziranih jezika, c i c++ takođe. Međutim, treba napraviti razliku da na primer u c programskom jeziku promenljive mogu biti cast-ovane u druge tipove ali da se one tada ne konvertuju već da se prilikom čitanja samo predpostavlja da su one zadatog tipa.  Statičko tipiziranje ne znači da moramo najpre da deklarišemo sve promenljive pre nego što ih koristimo - promenljive mogu biti pokrenute bilo gde ali programeri moraju da to učine pre nego odluče da ih iskoriste bilo gde programu.   |  | | --- | | 1 *// explicit declaration*  2 **static** int num, sum;  3 *// now use the variables*  4 num = 5;  5 sum = 10;  6 sum = sum + num; |   Dinamički tipizirani programski jezici su oni u kojima promenljive moraju deklarisati takođe pre njihovog korišćenja. Međutim, ovo podrazumeva da dinamički tipizirani jezici ne zahtevaju eksplicitnu deklaraciju promenljivih pre njihovog korišćenja. Python je jedan od primera dinamički tipiziranog programskog jezika kao i PHP.   |  | | --- | | 1 *# directly using the variable*  2 num = 10 |   Primer dinamičkog tipiziranja predstavljen je preko promenljive num kojoj je direktno bila dodeljena vrednost 10 pre inicijalizacije. Ovo je karakteristika dinamički tipiziranih programskih jezika.  Pojam dynamic odnosno **duck typing** dobilo je ovo ime po popularnoj izjavi – „if it walks like a duck and quacks like a duck its probably a duck“. Ovo takođe znači da ukoliko određena promenljiva izgleda i ponaša se kao string ili integer onda je ona verovatno string ili integer tipa.   |  | | --- | | 1 x = "hello"  2 puts x.class *# returns String*  3  4 x = 10  5 puts x.class *# returns Fixnum*  6  7 x = x.to\_f  8 puts x.class *# returns Float* |   **Strong/Weak typing**  Jako tipiziranje je najčešći model tipiziranja u modernim jezicima. Ovako tipizirani jezici imaju jednu jednostavnu karakteristiku - znanje o tipu vrednosti u vreme izvršavanja. Može se reći da jako tipizirani jezici sprečavaju mešanje operacija između tipova koji se ne podudaraju.   |  | | --- | | 1 String foo = "Hello, world!";  2 Object obj = foo;  3  4 String bar = (String) obj;  5 Date baz = (Date) obj; *// This line will throw an error* |   Ovaj primer će raditi perfektno sve do poslednje linije koda gde će biti bačen **ClassCastException** zbog toga što je java jako tipizirani jezik..  Kada govorimo o slabo tipiziranim jezicima, perl je jedan od njih. Sledeći primer ilustruje kako Perl nema nikakvih problema sa mešanjem različite vrste tipova.   |  | | --- | | 1 $a = 10;  2 $b = "a";  3 $c = $a . $b;  4 **print** $c; *# returns 10a* |   **Nominal/Structural typing**  Nominalno tipiziranje odnosno tipiziranje sistema bazirano na imenima je glavna klasa type sistema u kojoj se kompatibilnost i jednakost tipova podataka određuje eksplicitnim deklaracijama imenom tipova. Nominalni sistemi se koriste za određivanje jednakosti tipova kao i za proveravanje da li je tip podtip nekog drugog tipa.  Nominalno tipiziranje je u suprotnosti sa strukturalnim tipiziranjem bazirano na svojstvima type sistema koje predstavlja glavnu klasu type sistema a u kojoj se kompatibilnost i jednakost tipova podataka određuje na osnovu aktuelne strukture tipa ili definicije a ne na osnovu drugih karakteristika kao što je ime ili mesto deklaracije. Strukturalno tipiziranje se koristi kako bi se odredilo da li su određeni tipovi jednaki i da li je tip podtip nekog drugog.   |  | | --- | | 1 record PolarComplexNumber  2 {  3 double phase, magnitude;  4 };  5  6 record VelocityVector  7 {  8 double phase, magnitude;  9 }; |   U prethodnom primeru, kada govorimo o strukturalnom sistemu tipiziranja – dva tipa bi bila ekvivalentna s obzirom da imaju istu strukturu. Kada govorimo o nominalnom sistemu tipiziranja – dva tipa ne bi bila ekvivalentna s obzirom da poseduju različita imena (PolarComplexNumber vs VelocityVector).  Sledeći primer ilustruje dodavanje trećeg atributa (azimuth). Kada govorimo o strukturalnom sistemu tipiziranja – ovakav novi record predstavlja podtip VelocityVector tipa dok u nominalnom ne – kako bi dobio deklaraciju podtipa jedan od njih bi morao eksplicino da to deklariše.   |  | | --- | | 1 record VelocityVector3d  2 {  3 inherit VelocityVector;  4 double azimuth;  5 }; |   njih |
| **Naslov sekcije** |
| ***Smernice za pisanje koda*** |
| **Sadržaj (telo) sekcije** |
| Generalno, kada govorimo o skripting jezicima postoje neke smernice koje opisuju kako bi kod trebalo da izgleda u nekima od njih (Perl, Ruby, Python, itd.).  **Nivo indentacije**  Nivo indentacije (broj razmaka na početku linije) mora biti konzistentan. Ovo je posebno važno u Python-u, gde je indentacija deo sintakse jezika – uvučena linija označava početak novog bloka, a nekonzistentan broj razmaka pri uvlačenju rezultuje sintaksnom greškom, ili u opasnijim (i retkim) slučajevima, pogrešno izvršenim kodom.  Za Python se tipično koriste četiri karaktera za indentaciju. Ruby, s druge strane, tipično koristi dva karaktera. Za perl i javascript prakse nisu toliko ustaljene, ali je četiri najčešći broj.  **Tabs vs spaces**  Korišćenje znaka za razmak umesto tabova je generalno preporučeno, jer se tab karakter '\t' prikazuje drugačije zavisno od toga na kom medijumu se prikazuje. Dok samo korišćenje tabova nije preveliki problem, korišćenje izmešanih znakova razmaka i tabova jeste. Neki terminali imaju tab-stopove za prikazivanje tabova, dok neki drugi terminali (i većina tekst editora) tab tretira kao određeni broj karaktera razmaka (4 ili 8). Ovo znači da vaš lepo tabovima formatiran kod može izgledati potpuno rasformatirano na nekom drugom editoru. Većina modernih editora (nodepad++, Geany, Gedit, Sublime Text 2/3 i ostali) podržavaju pretvaranje tabova u razmake, i ovu mogućnost treba koristiti.  **Korišćenje globalnih promenljivih**  U većini programskih jezika je korišćenje globalnih promenljivih praksa koja se obeshrabruje, pa ni ova četiri jezika nisu izuzetak. Dok sami jezici koriste globalne promenljive (Perl i Ruby posebno), one su interne i deo okruženja koje taj jezik pruža. Globalne promenljive su obično deo globalnog stanja, koje bi trebalo enkapsulirati u posebne funkcije, metode, klase ili module.  **Konzistentnost**  Konzistentnost je jedna od osnovnih praksi pisanja kvalitetnog koda. Konzistentnost u ovom slučaju znači da treba izabrati jedan način (ili stil) pisanja koda, i pridržavati se izabranog u okviru celog projekta. Ovo uključuje odluke vezane za imenovanje promenljivih, nivo indentacije, položaj vitičastih zagrada i slično. Koji god (validan) stil izaberete – pridržavajte ga se.  Pored ovih generalnih, svaki jezik ima svoje posebne smernice. Uz ovo vežbanje možete naći linkove ka preporučenim praksama za svaki od ovih jezika. |

|  |  |
| --- | --- |
| **RELEVANTNI LINKOVI** | |
| **Opis** | **URL** |
| Preporuke za Python | http://www.python.org/dev/peps/pep-0008/ |
| Preporuke za Perl | http://www.perl.org/books/beginning-perl/ |
| Preporuke za Ruby | http://www.caliban.org/ruby/rubyguide.shtml |
| Preporuke za JavaScript | <http://javascript.crockford.com/code.html> |

|  |  |
| --- | --- |
| **REČNIK** | |
| **Termin** | **Opis** |
|  |  |

|  |
| --- |
| **DRUGI PREDLOŽENI SADRŽAJI AUTORA** |
| **Naslov** |
|  |
| **Telo** |
|  |