Lekcija 1 - Uvod u skripting jezike

Contents

LearningObject

L1: Uvod u skripting jezike

Uvod *
Ovo poglavlje predstavlja uvod u oblast skripting jezika.

Teorija i koncept programiranja

Cilj

- Upoznavanje sa teorijom jezika i konceptom programiranja

Teorija i koncept programiranja

Teorija jezika i koncept programiranja

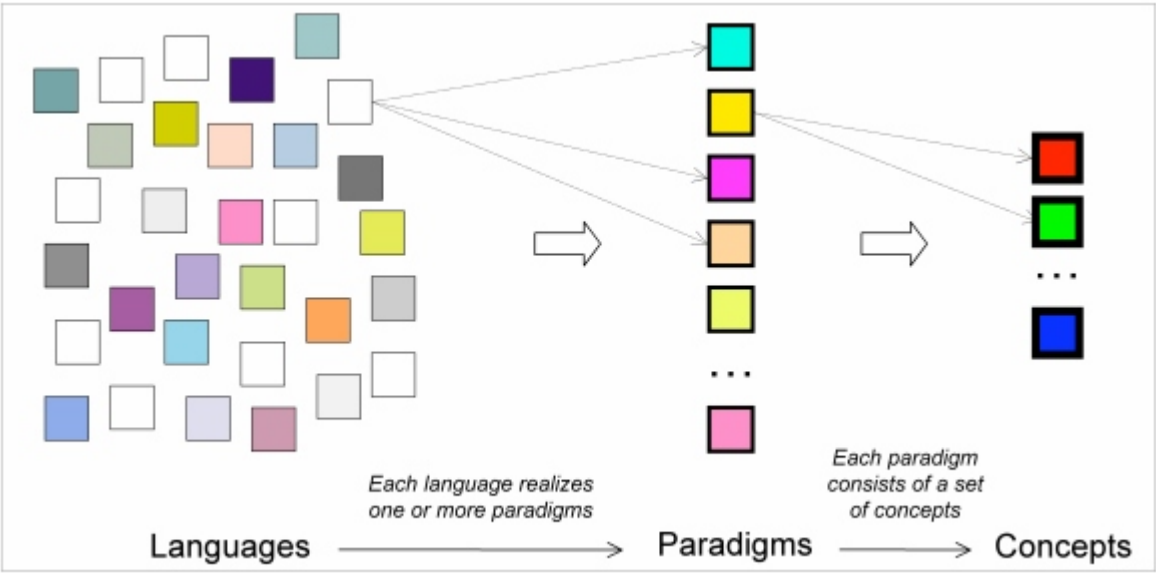
U teoriji jezika postoje razlike između:

1. funktionalnog i proceduralnog jezika
2. imperativnog i deklarativnog jezika
3. dinamičkog i statičkog jezika
4. jakog i slabog jezika

Programski jezik

Koncept (engl. paradigm) programiranja

Svaki jezik se realuzuje jednu ili više paradigmi, a svaka paradigma se sastoji od skupa koncepata, Slika 1.



Slika 1 Jezici, paradigme i koncepti

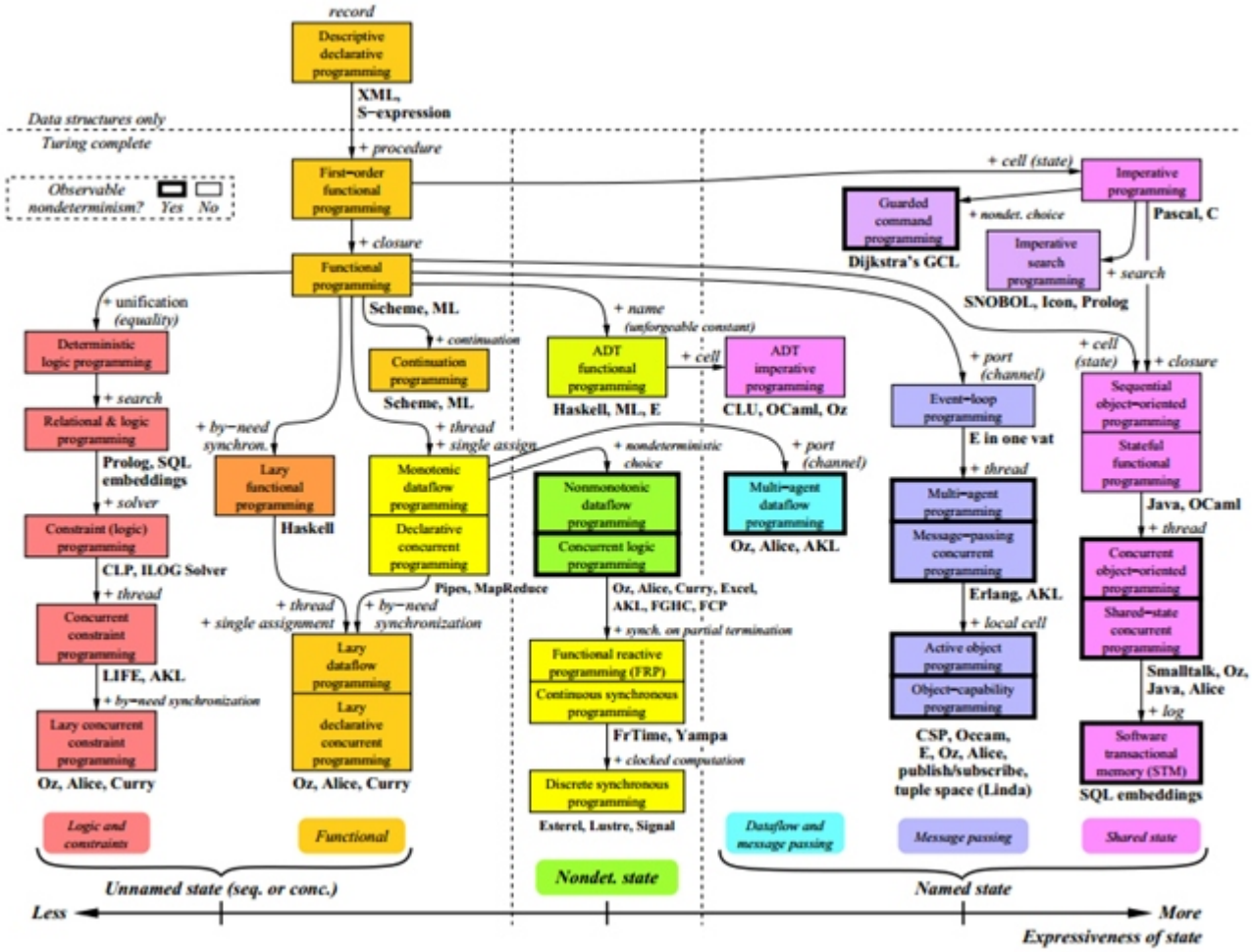
Programski jezik je način da se abstraktne karakteristike problema odvoje od realizaciej rješenja tog problema.

Koncept programskog jezika

Koncept programskog jezika je način da se oblikuje ova apstrakcija da odgovara određenom domenu problema. Konceptv programiranja je osnovni stil računarskog programiranja. Uglavnom oostoje četiri osnovna koncepta programiranja:

1. objektno-orijentisano programiranje
2. imperativno programiranje
3. funkcionalno programiranje
4. logičko programiranje

Slika 2 daje taksonomiju paradigmi programiranja.



Slika 2 Taksonomija paradigmi programiranja

Objektno-orijentisano programiranje

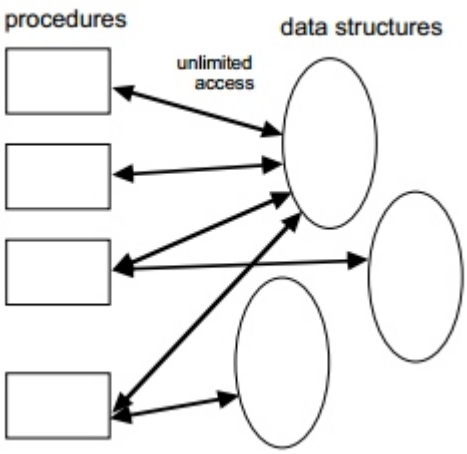
Cilj

- Upoznavanje sa konceptom OOP

Objektno-orijentisano programiranje

Objektno-orijentisano programiranje (engl. Object Oriented Programming OOP)

Slika 1 prikazuje tradicionalnu paradigmu programiranja.



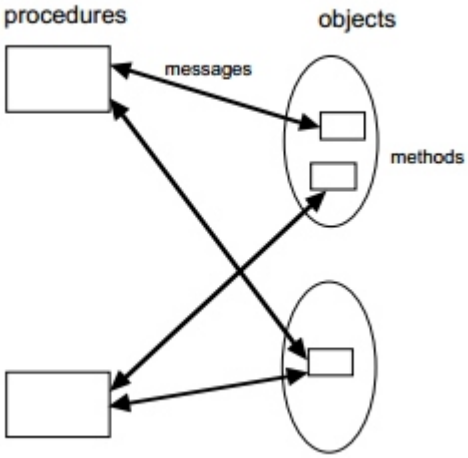
Slika 1 Tradicionalna paradigma programiranja

Objektno-orijentisano programiranje ili OOP je koncept programiranja koji koristi "objekte" tj. instance klasa>ove klase se sastoje od polja podataka i metoda zajedno sa njihovom interakcijom i služe za kreiranje aplikacija i računarskih programa.

Tehnike programiranja mogu uključiti karakteristike kao što su:

- abstrakcija podataka (engl. data abstraction)
- enkapsulacija, (engl. encapsulation)
- razmjena poruka (engl. messaging)
- modularnost (engl. modularity)
- polimorfizam (engl. polymorphism)
- nasleđivanje (engl. inheritance)

Slika 2 prikazuje OOP paradigm.



Dodatak

Object-Oriented Programming Concepts

If you've never used an object-oriented programming language before, you'll need to learn a few basic concepts before you can begin writing any code. This lesson will introduce you to objects, classes, inheritance, interfaces, and packages. Each discussion focuses on how these concepts relate to the real world, while simultaneously providing an introduction to the syntax of the Java programming language.

What Is an Object?

An object is a software bundle of related state and behavior. Software objects are often used to model the real-world objects that you find in everyday life. This lesson explains how state and behavior are represented within an object, introduces the concept of data encapsulation, and explains the benefits of designing your software in this manner.

What Is a Class?

A class is a blueprint or prototype from which objects are created. This section defines a class that models the state and behavior of a real-world object. It intentionally focuses on the basics, showing how even a simple class can cleanly model state and behavior.

What Is Inheritance?

Inheritance provides a powerful and natural mechanism for organizing and structuring your software. This section explains how classes inherit state and behavior from their superclasses, and explains how to derive one class from another using the simple syntax provided by the Java programming language.

What Is an Interface?

An interface is a contract between a class and the outside world. When a class implements an interface, it promises to provide the behavior published by that interface. This section defines a simple interface and explains the necessary changes for any class that implements it.

What Is a Package?

A package is a namespace for organizing classes and interfaces in a logical manner. Placing your code into packages makes large software projects easier to manage. This section explains why this is useful, and introduces you to the Application Programming Interface (API) provided by the Java platform.

Imperativno programiranje

Cilj

- Upoznavanje sa konceptom iImperativnog programiranja

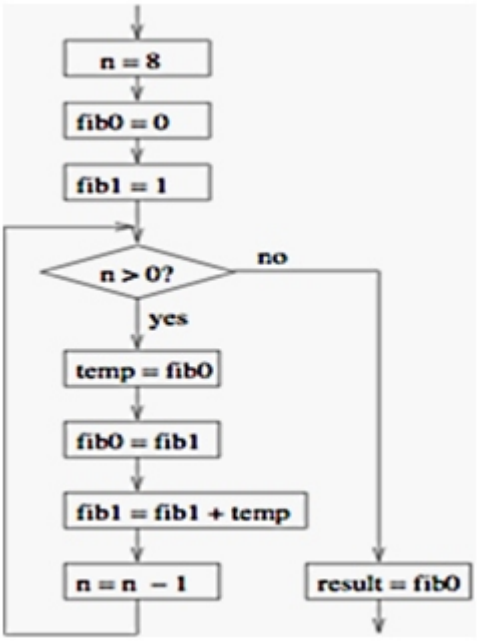
Imperativno programiranje

Imperativno programiranje

Imperative programiranje je koncept programiranja koji opisuje process računanja uz pomoć sekveence izjava (engl. statement) koje mijenjaju stanje programa.

Kod imperativnog programiranja, program čini niz instrukcija računara. Osnovni alat koji omogućava ovaj način programiranja je uslovni skok.

Dijagrami toka (engl. flow dijagram) su običnokoriste kod imperativnog programiranja, Slika 1.



Slika 1 Dijagram toka

Primer imperativnog koda

```
var numbers = [1,2,3,4,5]
var doubled = []

for(var i = 0; i < numbers.length; i++) {
var newNumber = numbers[i] * 2
doubled.push(newNumber)
}
console.log(doubled) //=> [2,4,6,8,10]
```

Dodatak

Imperative programming *is a programming paradigm that describes computation in terms of statements that change a program state. In much the same way that imperative mood in natural languages expresses commands to take action; imperative programs define sequences of commands for the computer to perform. The term is used in opposition to declarative programming, which expresses what the program should accomplish without prescribing how to do it in terms of sequences of actions to be taken****. Functional and logic programming*** *are examples of a more* ***declarative*** *approach.*

Logic programming *is a programming paradigm based on formal logic. Programs written in a logical programming language are sets of logical sentences, expressing facts and rules about some problem domain. Together with an inference algorithm, they form a program. Major logic programming languages include Prolog and Datalog.*

Functional programming *is a programming paradigm, a style of building the structure and elements of computer programs that treats computation as the evaluation of mathematical functions and avoids state and mutable data. Functional programming emphasizes functions that produce results that depend only on their inputs and not on the program state—i.e. pure mathematical functions. It is a declarative programming paradigm, which means programming is done with expressions. In functional code, the output value of a function depends only on the arguments that are input to the function.*

Proceduralno programiranje

Cilj

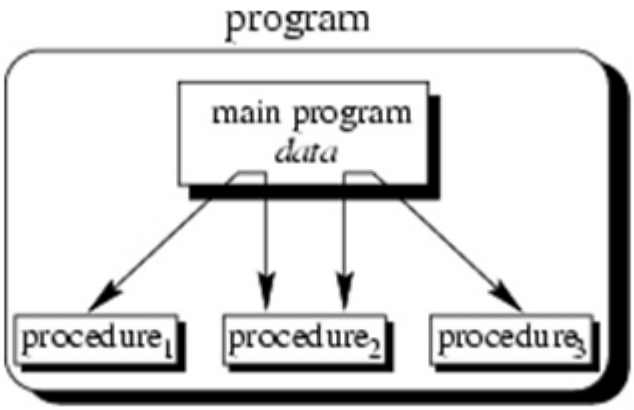
- Upoznavanje sa proceduralnim programiranjem

Proceduralno programiranje

Proceduralno programiranje

Ovaj način programiranja se često može uzeti kao sinonim za imperativno programiranje (definisanje koraka koje program mora proći da bi došao u željeno stanje).

Slika 1 ilustruje proceduralno programiranje.



Slika 1Procedurano programiranje

Takođe se može nazvati i konceptom programiranja koji je izvedeno od strukturnog programiranja (engl. structured programming) koje je bazirano na konceptu poziva procedura (engl. procedure call).

Strukturno programiranje

Strukturno programming je koncept programiranja s ciljem da se poboljša jasnoća, kvalitet i vrijeme razvoja računarskog programa uz često korišćenje potprograma, blokova, and “for” i “while” petlji umjesto korišćenja jednostavnih testova i skokova u programu kao što su the “go to” naredbe.

Procedura koja se takođe naziva i rutinom (engl. routine), potprogramom (engl. subroutines), metodom, ili funkcijom (ne treba miješati sa matematičkim funkcijama koje se koriste u funktionalnom programiranju), je serija koraka koje zahtijevaju operacije računanja.

Bilo koja procedura može biti pozvana iz bilo kojeg dijela programa u toku njegovog izvršavanja, pozivom druge procedure i pozivom samog sebe. Ovdje se uvodi ideja poziva procedure koja uvodi definisanje lokalnih varijabli u ovim procedurama. Time se ograničava sposobnost drugih procedura da pristupe lokalnim varijablama.

Proceduralni jezici su:

- C
- Java
- Perl

Rezime: Proceduralno programiranje

Računarski program je mašina stanja (engl. state machine)

Varijable mogu mijenjati stanja

Dodatak

Procedural programming*can sometimes be used as a synonym for imperative programming (specifying the steps the program must take to reach the desired state), but can also refer (as in this article) to a programming paradigm, derived from structured programming, based upon the concept of the procedure call. Procedures, also known as routines, subroutines, methods, or functions (not to be confused with mathematical functions, but similar to those used in functional programming) simply contain a series of computational steps to be carried out. Any given procedure might be called at any point during a program's execution, including by other procedures or it.*

Procedures and modularity

Modularity*is generally desirable, especially in large, complicated programs. Inputs are usually specified syntactically in the form of arguments and the outputs delivered as* ***return values****.*

Scoping*is another technique that helps keep procedures strongly modular. It prevents the procedure from accessing the variables of other procedures (and vice-versa), including previous instances of itself, without explicit authorization.*

Less modular procedures, often used in small or quickly written programs, tend to interact with a large number of variables in the execution environment, which other procedures might also modify. Because of the ability to specify a simple interface, to be self-contained, and to be reused, procedures are a convenient vehicle for making pieces of code written by different people or different groups, including through programming libraries.

Comparison with imperative programming

Procedural programming languages are also imperative languages, because they make explicit references to the state of the execution environment. This could be anything from variables (which may correspond to processor registers) to something like the position of the "turtle" in the Logo programming language.

Deklarativno programiranje

Cilj

- Upoznavanje sa konceptom deklarativnog programiranjanja

Deklarativno programiranje

Deklarativno programiranje

Za razliku od Imperativnog programiranja, deklarativno programiranje,izražava ono što program treba da ostvari bez propisivanja kako to učiniti u smislu sekvence akcija koje treba poduzeti.

Primer deklarativnog koda

var numbers **=** [1,2,3,4,5]

var doubled **=** numbers.mapv(**function**(n) { **return** n * 2})

console.log(doubled) //=> [2,4,6,8,10]

Ovdje map kreira novi niz od postojećeg niza pri čemu se svaki element novog niza kreira slanjem elemenata starog niza u neku funkciju ako argument. Ovdje je funkcija map (function(n) { return n*2 }).

Deklarativno programiranje može biti :

- Funkcionalno programiranje
- Logičko programiranje

Funkcionalno programiranje

Ovaj concept programiranja tretira računanje kao evaluaciju matematičkih funkcija i izbjegava stanja i promjenjive podatke. Fokusira se na primjeni funkcija (za razliku od od imperativnog programskog stila, koji naglašava promjene stanja).

Funkcionalna programiranje ima svoje korijene u lambda kalkulusu (engl. lambda calculus), formalnom sistemu razvijenom u 1930 za istraživanje načini definisanja funkcija, aplikacije funkcija i rekurzije.

Neki functionalni jezici su:

- Idris
- ML
- Haskell

Rezime: Funkcionalno programiranje

- Objekti su konstante ili funkcije
- Red izvršavanja operacija nije soecificiran
- Ne definiše stanja (engl. state)

Logičko programiranje

Logičko programiranje se bazira na ideji da se koriste logički izrazi za predstavljanje programa i za izvršavanje računskih operacija. Način logičkog izraza koji se može naći u logičkom programiranju je definisan u obliku sledećege izraza:

G if G1 and … and Gn

Dinamičko programiranje

Cilj

- Upoznavanje sa dinamičkim programiranjem

Dinamičko programiranje

Dinamičko programiranje

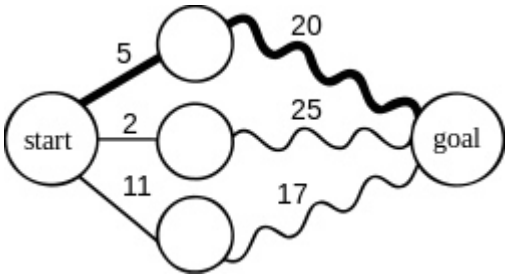
Dinamičko programiranje je termin koji se koristi za opisivanje klase programskih jezika visokog nivoa (engl. high-level programming languages) koji u toku svog izvršavanja (engl. runtime) ispoljavaju mnogo osobina koje drugi jezici mogu obavljati u toku svog prevođenja ili kompilacije.

Ovo ponašanja može uključivati proširenje datog programa, dodavanje novog kôda, širenje sadržaja objekata i definicije, ili mijenjanjem sistema tipiziranja i to sve u vrijeme izvršavanja programa.

Ovo se ponašanja može emulirati u bilo kojem jeziku dovoljne kompleksnosti, ali dinamički jezici pružaju direktne alate za neposredno korišćenje.

Dinamičko programiranje se može koristiti za rješavanje složenih problema tako da se razbiju na jednostavnije podprobleme. To se odnosi na probleme koji pokazuju svojstva preklapanja podproblema (engl. overlapping subproblems)i optimalno podstrukture (engl. optimal substructure).

Kada se metod može primeniti, uzima daleko manje vremena nego naivne metode koje ne koriste prednosti podproblema (na primer depth-first search).



Slika 1 Traženje najkraće putanje

Dodatak

The idea behind dynamic programming is quite simple. In general, to solve a given problem, we need to solve different parts of the problem (subproblems), then combine the solutions of the subproblems to reach an overall solution. Often when using a more naive method, many of the subproblems are generated and solved many times. The dynamic programming approach seeks to solve each subproblem only once, thus reducing the number of computations: once the solution to a given subproblem has been computed, it is stored or "memo-ized": the next time the same solution is needed, it is simply looked up. This approach is especially useful when the number of repeating subproblemsgrows exponentiallyas a function of the size of the input.

Sistem tipiziranja

Cilj

- Upoznavanje sa sistemom tipiziranja

Sistem tipiziranja

Sistem tipiziranja (engl. type system)

Ovaj system pridružuje tip za svaku izračunatu vrijednost. Sistem ispituje tok ovih vrijednosti i tako pokušava da osigura ili pokaže da nema grešaka tipova (engl. type errors). Slika 1 prikazuje karakteristike sistema tipiziranja

Sistemi tipiziranja
Type safety
Dynamic i Static
Strong i Weak
Nominal i Structured
Duck typing
Linear typing

Slika 1 Karakteristike sistema tipiziranja

Formalno, teorija tipizitranja (engl. type theory) proučava sisteme tipiziranja (engl. type system). Jezik programiranja mora vršiti provjeru tipiziranja (engl. type check) koristeći sisteme tipiziranja bilo u vrijeme prevođenja programa (engl. compiler time) ili u vrijeme izvršavanja programa (engl. runtime).

Bezbjednost tipova podataka (engl. type safety)

Bezbjednost tipova znači mjera u kojoj programski jezik obeshrabruje ili spriječava greške tipiziranja.

Bezbjednost tipova je nekad karakteristika računarskog program, a ne jezika u kojem je pisan program [8], [9].

Greška tipova

Greška tipova je pogrešno ili nepoželjno ponašanje uzrokovano razlikama između različitih tipova podataka koji se koriste za programske konstante, varijable, i metode (funkcije), na pr. tretiranje tipa integer (int) kao floating point tip (float).

Određeni sistem tipova tačno određuje šta predstavlja grešku tipa, ali cilj je spriječiti operacije kao što su logičke greške ili memorijske greške. Sistem tipova je obično dio programskog jezika ugrađen u interpretere i prevodioce mada mogu biti implementirani i kao opcioni alati.

Tipiziranje podataka (engl. typing) može biti slabo tipiziranje podataka (engl. weak typing) i jako tipiziranje podataka (engl. strong typing).

Bezbedni jezici

Cilj

- Upoznavanje sa bezbednim jezicima

Bezbedni jezici

Bezbedni jezici (engl. safe languages)

Programski jezik je bezbjedan ako ne dozvoljava operacije koje mogu da generisati grešku i izazovu grešku.

Primjer: bezbedni jezik

var x = 5;

var y = "hi";

x + y; // 5hi

Primjer: nebezbjedni jezik

var x := 5;

var y := "37";

var z := x + y;

Rezultat: 42

Osnovni atribut sistema za tipiziranje je pitanje nominativnog tipiziranja (engl.nominative typing) i strukturnog tipiziranja (engl. structural typing).

Dodatak

http://researcher.watson.ibm.com/researcher/view_project.php?id=1597

Language-Based Securityis the area of research that studies how to enforce application-level security using programming-language and program-analysis techniques.LaBaSecis the Language-Based Security project at the IBM T. J. Watson Research Center. The purpose of LaBaSec is to automate the detection of access-control and information-flow vulnerabilities in software due to coding malpractice or security-policy misconfigurations, study the design and implementation of secure programming languages, promote the correct usage of security Application Programming Interfaces-APIs, and enforcing Digital Rights Management-DRM.

Software security has been traditionally enforced at the level of operating systems. However, operating systems have become increasingly large and complex, making it very difficult, if not impossible, to enforce software security solely through them. Moreover, operating-system security allows dealing primarily with access-control policies on resources such as files and network connections, while attacks may happen at both lower and higher levels of abstraction, and may target the internal behavior of applications. Therefore, defenses must offer protection at the level of applications.

LaBaSec focuses on the following areas:

Building program analysis tools for automatic identification of access-control and flow violations

Many common security flaws can be discovered through static analysis of software code and artifacts. The code of a program can either be in source or binary format. The software artifacts associated with a program may include policy databases, configuration files, deployment descriptors and other metadata that describes the program and its intended operation or security characteristics.

Designing and implementing new programming languages that incorporate security features for better definition and enforcement of access-control and information-flow policies.

New programming languages designed with security in mind embed support for security features, such as definition, tracking and enforcement of information-flow and access-control policies.

Building static-analysis tools to enforce correct usage of security APIs.

Modern run-time environments, such as theJava Runtime, Microsoft .NET Common Language Runtime-CLR and PHP, offer a number of security APIs for cryptography, access control and information flow. Unfortunately, these APIs are often difficult to use and not well documented. Misusing a security API is something that a compiler or a run-time interpreter may not necessarily catch, and can expose a program to serious security holes

Transparently enforcing Digital Rights Management.

In the past few years, a number of DRM products have been designed and created that attempt to address the issue of licensing and controlling the distribution of digital contents. In general, the consumer of the digital content is required to install a customized client-side DRM-enabled player. Such a player verifies and enforced the digital rights that a user has acquired. This approach is somewhat limited since it requires the installation of DRM-enabled players on the client system.

Tipiziranje

Cilj

- Upoznavanje sa konceptom tipiziranja

Tipiziranje
Tipiziranje (engl. typing)

Tipiziranje kod prograskih jezika znači, klasifikacija varijabli prema tome kakav podatak sadrže (na pr. string, integer, floating point).

Jako tipizirani jezici (engl. strongly typed languages) sprovode strogo pridržavanje tipiziranju i ne dopuštaju miješanje raznih podataka u istoj varijabli.

Slabo tipizirani jezici (engl. weakly typed languages) osiguravaju minimalnu provjeru, što može dovesti do grešaka u obradu podataka.

Provjera tipa (engl. type checking)

Proces provjere i provođenje ograničenja vezano za tipizaciju - provjera tipa – se može se pojaviti pri prevođenja programa vremenu (engl. compile time) što predstavlja statičku provjeru ili u toku izvršavanja programa (engl. run-time) što predstavlja dinamička provjeru.

Slabo tipiziranje (engl. weak typing)

Kod ovog metoda, variables se interpretiraju kao da imaju više tipova. Programski jezik automatski menja tip da bi data operacija bila uspješna.

Primjeri: Visual Basic

```
var x = 5;

var y = "hi";
```

x + y; // 5hi

Jako tipiziranje (engl. strong typing)

Kod ovoga metoda, varijables imaju samo jednu vrijednost. Programski jezik ne dozvoljad da operacija bude uspješna ako su tipovi pogrešni.

Primjer: Python

v = ’1.2’;

x = 5 * b; # runtime error

Slika 1 poredi jako i slabo tipiziranje jezike.

	Weak Typing	Strong Typing
Pseudocode	a = 2 b = "2" concatenate(a, b) # Returns "22" add(a, b) # Returns 4	a = 2 b = "2" concatenate(a, b) # add(a, b) # concatenate(str(a), b) # add(a, int(b)) #
Languages	BASIC, JavaScript, Perl, PHP, Rexx	ActionScript 3, C++, C#, Java, Pyth

Slika 1 Jako i slabo tipiziranje

Dinamičko tipiziranje (engl. dynamic typing)

Kod ove metode, varijable mogu da imaju različite tipove podataka. Prevodilac zna koji je korektan tip podatka.

Primjeri: Python

c = 1 # an integer

c = “a string” # a string

c = [a, b, c] # a list

“Duck” tipiziranje

Ovaj metod je tip dinamičkog tipiziranja gdje trenutni skup metoda i svojstava nekog objekta određuje korektnu semantiku, a ne naslijeđene karakteristika neke određene klase. Drugim riječima, nekoj varijabli se dodjeljuje tip zavisno od njenog sadržaja.

Primjer

function calculate(a, b, c) => return (a+b)*c

example1 = calculate (1, 2, 3)

example2 = calculate ([1, 2, 3], [4, 5, 6], 2)

example3 = calculate ('apples ', 'and oranges, ', 3)

Statičko tipiziranje podataka (engl. static typing)

Kod ovog metoda, varijable mogu da imaju samo jedan tip. U toku izvršavanja program, mora se odrediti pravi tip podataka.

Primjeri: C

double c;

c = 5.2;

c = “A string”;

Nominativno tipiziranje (engl. nominative typing)

Varijable su kompatibilne samo ako imaju iste tipove.

Primjer: Java

public int add (int a, int b) {

return(a + b);

}

Strukturno tipiziranje (engl. structural typing)

Varijable su kompatibilne ako imaju istu strukturu.

Primjer: Java

record PolarComplexNumber

{

double phase, magnitude;

};

record VelocityVector

{

double phase, magnitude;

};

Slika 2 poredi programske jezike.

Programming language	Static / Dynamic	Strong / Weak	Safety	Nominative / structural
Ada	Static	Strong	Safe	Nominative
assembly language	None	Weak	Unsafe	Structural
BASIC	Static	Weak	Safe	Nominative
C	Static	Weak	Unsafe	Nominative
C++	Static	Weak	Unsafe	Nominative
C#	Static	Strong	Safe	Nominative
Haskell	Static	Strong	Safe	Nominative
Java	Static	Strong	Safe	Nominative
Lisp	Dynamic	Strong	Safe	Structural
ML	Static	Strong	Safe	Structural
Objective-C	Dynamic	Weak	Safe	Duck
Perl	Dynamic	Weak	Safe	Nominative
PHP	Dynamic	Strong	Safe	Nominative
Python	Dynamic	Strong	Safe	Duck
Ruby	Dynamic	Strong	Safe	Duck
Scheme	Dynamic	Weak	Safe	Nominative
Smalltalk	Dynamic	Weak	Safe	Duck

Slika 2 Poređenje programskih jezika

Tipovi podataka

Cilj

- Upoznavanje sa tipovima podataka

Tipovi podataka

Tipovi podataka (engl. data types)

Tip podataka je klasifikacija koja identifikuje jedan od raznih vrsta podataka, kao što je real, integer, ili Boolean, koji određuje moguće vrijednosti za taj tip, a operacija koje se može obaviti na vrijednosti tog tipa; značenje podataka; i način na koji se vrijednosti tog tipa mogu biti sačuvani.

Većina vrsta podataka u statistici ima uporedive vrste u računarskom programiranju i obrnuto, kao što je prikazano na Slici 1.

Statistics	Computer programming
real-valued (interval scale)	floating-point
real-valued (ratio scale)	
count data (usually non-negative)	integer
binary data	Boolean
categorical data	enumerated type
random vector	list or array
random matrix	two-dimensional array
random tree	tree

Slika 1 Vrste podataka u programiranju

Tipični tipovi podataka mogu da budu:

- integer
- boolean
- character
- floating-point number
- alphanumeric strings.

Na primjer, vrednosti bilo kojeg tipa podataka mogu se sačuvati u varijabli Perl jezika, Slika 2.

```
$myVar = 'c'; # Character
$myVar = "Hello World!"; # String
$myVar = 42; # Integer
$myVar = 3.14159; # Float
```

Slika 2 Perl primer

Slika 3 ilustruje kako se automatski vrši konverzija iz jednog tipa podatka drugi u Perl jeziku.

From	To	Conversion
"42"	42	String to integer
42	"42"	Integer to string
"3.14159"	3.14159	String to float
3.14159	"3.14159"	Float to string
"c"	'c'	String to char
'c'	"c"	Char to string

Slika 3 Konverzija tipova podataka u Perlu

Dodatak

Data type*or simply* ***type*** *is a classification identifying one of various types of data, such as as real, integer or Boolean, that determines the possible values for that type; the operations that can be done on values of that type; the meaning of the data; and the way values of that type can be stored*

Definition of a "type"

In 1976 Parnas, Shore and Weiss identified five definitions of a "type" that were used—sometimes implicitly-in the literature:

Syntactic

A type is a purely syntactic label associated with a variable when it is declared. Such definitions of "type" do not give any semantic meaning to types.

http://en.wikipedia.org/wiki/Syntax

http://en.wikipedia.org/wiki/Semantic

Representation

A type is defined in terms of its composition of more primitive types-often machine types.

Representation and behavior

A type is defined as its representation and a set of operators manipulating these representations.

Value space

A type is a set of possible values which a variable can possess. Such definitions make it possible to speak about (disjoint) unions or Cartesian products of types.

Value space and behavior

A type is a set of values which a variable can possess and a set of functions that one can apply to these values.

The definition in terms of a representation was often done in imperative languages such as ALGOL and Pascal, while the definition in terms of a value space and behavior was used in higher-level languages such as Simula and CLU.

Skriptnig jezici

Cilj

- Upoznavanje sa skriptnig jezicima

Skriptni jezici

Skriptni jezici

Definicija

Skriptni jezik je programski jezik koji podržava pisanje skripti, programa napisanih za softverska okruženja koja automatizuju izvršavanje zadataka, a koje bi operatorr inače morao da izvršava korak po korak.

Okruženja koje se mogu automatizovati uz pomoć skripti uključuju softverske aplikacije, web stranice unutar web pretraživača, ljuske operacionih sistema i nekoliko programskih jezika opšte namjene.

Skriptni jezici se razlikuju od programskim jezika po tome [to su napravljeni za "lijepljenje" aplikacija zajedno. Oni koriste “typeless” pristup kako bi se postigao veći nivo programiranja i brži razvoj aplikacija u odnosu na programske jezike.

Primjer lepljenja

Slika 1 primer lepljenja.

```
# wl - run word count, then print (lpr) a set of files
#
# Usage: wl file1 [file2] ...
T=/tmp/wl.$$        # temporary file
wc -l $* > $T       # get line counts for files
lpr $T $*           # print line counts, then files
rm $T               # remove temporary file
```

Slika 1 Primer lepljenja

U ovom prmjeru wl script „lijepi“ zajedo tri Unix alata wc, lpr, and rm. Svaki od njih može pojedinačno biti još jedan skript. Na ovaj način je kreiran koristan alat .

Skripta se može biti napisati i izvršiti "on-the-fly", bez eksplicitnih koraka prevođenje (engl. compile) i povezivanja (engl. link).

Pojam skripta obično je rezervisan za male programe (do nekoliko hiljada linija koda).

Karakteristike skripting jezika uključuju:

- Koriste ze obično za administraciju sistema i brzu izradu prototipova (engl. rapid prototyping)
- Skripting dozvoljava grupisanje tipično korišćenih komandi u batch datoteka za procesiranje
- Kreiranje novih fleksibilnih i konfigurabilnih alata koji „razumiju“ skripte i alate drugih korisnika
- Neformalan što se tiče tipiziranja varijabli,tj. nema razlike između tipova kao što su integer, floating-point ili string varijabla
- Funkcije mogu vratiti nonskalare , kao to nizovi (engl. array), neskalari se mogu koristiti kao ideksi petlji
- Mnogo operacija visikog nivoa ugrađene u dati jezik, na pr. push/pop.instrukcije
- Više interpretirani jezici nego kompilirani u transformaciji u skup mašinskih instrukcija na host mašini
- Skriptovi izbjegavaju manje greške nedosljednosti grešaka u procesiranju
- Tipični skripting jezici ukjlučuju ljuske skripta (sh, bash, csh, tcsh) i druge skripte (TCL, Perl, Python
- Ponovljeni zadatke može obaviti puno brže

Glavni nedostatak skriptnih jezika je da se izvršni kod skripta može i slučajno preuzeti s udaljenog servera na mašinu web pretraživača, instalirati i pokrenuti pomoću interpretera lokalnog pretraživača . To je lako učiniti tako da posjetite sumnjive web stranice ili preuzimanjem programa bez valjane autentičnosti. Korisnik vjerojatno nije svjestan svega što može dogoditi.

Skripting na strani klijenta

Cilj

- Upoznavanje sa skriptingom na strani klijenta

Skripting na strani klijenta

Skripting na strani klijenta (engl. client-side scripting)

When scripting languages found their way to the Internet world, they were divided into two parts-– Client side scripting and Server side scripting. Client side scripting languages are scripts, which are executed in the client's browser.

Kada su skriptni jezici pronašli svoj put do Interneta svijeta, oni su bili podijeljeni u dvije grupe:

- Skripting na strani klijenta
- Skripting na strani servera

Skripting jezici na na strani klijenta su skripte, koje se izvršavaju u klijentovom pretraživaču. Neki od najpopularnijih su: HTML, CSS (engl. Cascading Style Sheets), koji omogućuje da se metainformacije o stilu odvojen od sadržaja, XML, koji se općenito koriste samo za pohranu podataka i Java skripte također poznat kao emacs.

Važna stvar o ovom tipu skripti je da je njihov izvorni kod vidljiv svima što je mnogo pomoglo programerima početnicima u njihovim prvim koracima.

Skripting na strani servera

Cilj

- Upoznavanje sa skriptingom na strani servera

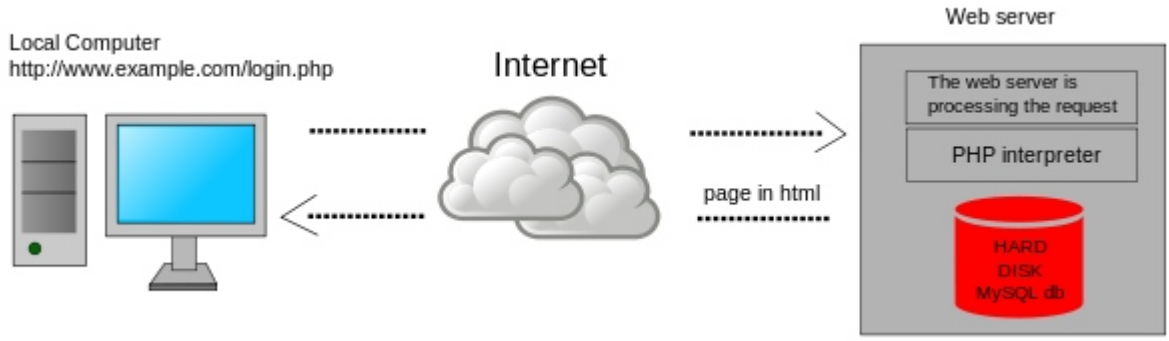
Skripting na strani servera

Skripting na strani servera (engl. server-side scripting)

Kada skripta naziva "server-side script", to znači da se izvršava na serveru i posjetilac nekog sajta može da vidi samo rezultat izvršavanja skripta.

To omogućava da se jako složeni skripti mogu koristiti, jer server je posvećen njihovom izvršenju. To također omogućava povezivanje skripte s bazama podataka i korišćenje podataka dok se skript izvršava. Popularni server-side skripte su PHP, Python, Perl, itd.

Slika 1lustruje scripting na strani servera.



Slika 1 Skripting na strani servera

Dodatak

Server-side scriptingis aweb servertechnology in which a user's request is fulfilled by running a script directly on the web server to generatedynamic web pages. It is usually used to provide interactive web sites that interface to databases or other data stores. This is different fromclient-side scriptingwhere scripts are run by the viewing web browser, usually inJavaScript. The primary advantage to server-side scripting is the ability to highly customize the response based on the user's requirements, access rights, or queries into data stores.

From security point of view, server-side scripts are never visible to the browser as these scripts are executes on the server and emit HTML corresponding to user's input to the page.

When the server serves data in a commonly used manner, for example according to theHTTPorFTPprotocols, users may have their choice of a number of client programs (most modern web browsers can request and receive data using both of those protocols). In the case of more specialized applications, programmers may write their own server, client, and communications protocol that can only be used with one another.

Programs that run on a user's local computer without ever sending or receiving data over a network are not considered clients, and so the operations of such programs would not be considered client-side operations

Vrste skripting jezika

Cilj

- Upoznavanje sa vrstama skripting jezika

Vrste skripting jezika
Vrste skripting jezika

Postoji više vrta scripting jezika. Neki od njih su

1. Jezik za upravljanje poslovima (engl. job control language) i ljuske (engl. shell)
2. GUI skripting jezici
3. Aplikacijski specifični jezici
4. Web pretraživači
5. Jezici za procesiranje teksta
6. Dinamički jezici opšte namjene
7. Ekstenzije/ugrađeni jezici (engl. embeddable languages)

Jezik za upravljanje poslovima

Cilj

- Upoznavanje sa jezikom za upravljanje poslovima

Jezik za upravljanje poslovima

Jezik za upravljanje poslovima (engl. job control language) i ljuske (engl. shell)

Glavni klasa skriptnih jezika je nastala iz iz automatizacije upravljanja poslovima što se odnosi na pokretanje i kontrolu ponašanja sistemskih programa.

Job Control Language -JCL je skriptni jezik za IBM mainframe operativni sistem koji upućuje sistem kako da pokrene paketni posao (engl. batch job) ili da pokrene podsistem.

Mnogi od interpretera tih jezika koriste se kao kao interpreteri komandne linije (engl. command-line interpreters) kao što su Unix lshell ljuska ili MS-DOS COMMAND.COM. Drugi skript jezici kao što je AppleScript koriste komande slične engleskom jeziku za za izgradnju skripti.

JCL je imao veliki uticaj na UNIX ljuske (engl. shells), bash, csh, ksh, and sh i Apple MPW. Korisnici ovih ljuski mogu izdavati komande sistemu za pomjeranje datoteka, kompiliranje programa i sl. Na primejr korisnik UNIX OS može izdati ove naredbe, Slika 1:

```
% mv foo.c bar.c        # rename "foo.c" to "bar.c"
% cc -o bar bar.c       # compile "bar.c" into "bar"
% bar
```

Slika 1 Naredbe UNIX ljuske

Linija između unosa komandi i skriptinga jasna.U većini modernih računarskih sistema proces snimanja „skripta“ korisničnih komandi je trivijalno. Ovaj se skript može mijenjati i ponovno koristiti. Tipični UNIX "shell script" za sortiranje je dat na Slici 2.

```
T=/tmp/wl.$$          # temporary file
wc  -l $* > $T        # get line counts for files
lpr $T $*             # print line counts, then files
rm  $T                # remove temporary file
```

Slici 2 UNIX shell skript

Skripti ljuske se mogu laše pisati, razumjeti i modifikovati, na primjer u odnosu na C programe kod izvršavanja istih zadataka. Mnkogi od standardnih UNIX alata su pisani kao skripting ljuske.

GUI skripting jezici

Cilj

- Upoznavanje sa GUI skripting jezicima

GUI skripting jezici

GUI skripting jezici

Uvođenjem GUI koncepta (engl. graphical user interface), pojavila se posebna vrsta skripting jezika za upravljanje računarom. Ovi jezici su u interakciji za grafičkim prozorima (engl. graphic windows), menijima, tasterima (na mišu ili tastaturi) itd. Ovi skriptni jezici se tipično koriste za simulaciju automatizaciju korisničkih komandi i obično se zovu makroi (engl. macros).

Ovi jezici se teoretski mogu koristiti za upravljanje GUI aplikacija; ali u praksi je njihovo korišćenje limitirano radi toga što zahtijevaju podršku i aplikacije i operativnog sistema.

Postoje nekoliko izuzetaka. Neki GUI skripting jezici se baziraju na prepoznavanju grafičkih objekata na ekranu. Ovi GUI skripting jezici ne zavise od podrške operativnog sistema ili aplikacije.

Aplikacijski specifični jezici

Cilj

- Upoznavanje sa aplikacijski specifičnim jezicima

Aplikacijski specifični jezici

Aplikacijski specifični jezici

Mnogi veliki aplikacijski programi uključuju skriptni jezik prilagođen potrebama korisnika aplikacije. Isto tako, mnogi računari prilagođeni računarskim igrama, koriste posebno kreirane skriptne jezike. Jezici ove vrste su kreirani za jednu aplikaciju mada površno gledano liče na neke jezike opšte namjene.

1. Web pretraživači

Web pretraživači su aplikacije za prikazivanje web stranica. Skripte mogu pokrenuti web pretraživače za promjenu izgleda ili ponašanja web stranice. Na primjer, mogu promijeniti sadržaj web stranice za specifičnog, trenutnog korisnika. Host ovih jezika specijalne namjene je razvijen da upravlja radom web pretraživača. To uključuje JavaScript; VBScript firme Microsofta, koji radi samo u Internet Explorer okruženju.

Xul firme Mozilla projekt radi samo u Firefoxu okruženju. XSLT jezik za prezentacije pretvara XML sadržaj u nove formate. Server šalje klijentove skripte onakve "kakve jesu" i klijent računar ih pokreće. Primjer skripte klijenta je JavaScript upozorenje (engl. alert box) koje iskače kada korisnik klikne na web stranici.

2. Jezici za procesiranje teksta

Obrada tekstualnih zapisa je jedan od najstarijih korišćenja skriptnih jezika. Skripte pisana za Unix alate awk, sed, i and automatizuju zadatke koji uključuju tekstualne datoteke, na primjer, konfiguracija i log datoteke.

Perl je izvorno dizajniran za prevladavanje ograničenja tih alata i posato je jedan od najraširenijih jezika opšte namjene.

3. Dinamički jezici opšte namjene

Neki jezici, kao što su Perl, počeli su kao skriptni jezici, ali su brzo razvili u programske jezike pogodnih za šire namjene. Ostali slični jezici - često interpretirani, memorija-uspio, ili dinamički - su opisani kao "skriptni jezici" zbog tih sličnosti i pored toga što se češće koristi za aplikacijsko programiranje.

4. Ekstenzije/ugrađeni jezici (engl. embeddable languages)

Odeđen broj jezika je bio kreiran za zamjenu aplikaciono specifičnih skriptnih jezika ugrađivanjem u aplikacione programe. Programer aplikacija (radeći u C ili drugom sistemskom jeziku) uključuje "veze" odakle skriptni jezik može kontrolirati aplikaciju.

Ovi jezici mogu biti tehnički ekvivalentni aplikaciono specifičnim jezicima, ali kada je u aplikaciju ugrađen jezik opšte, zejedničke namjene, korisnik dobiva prednost znog mogućnosti prenosa znanja i vještina od primjene do primjene.

JavaScript se počeo primarno koristiti kao jezik za skripring unutar web pretraživača, međutim, standardizacija jezika kao ECMAScript ga je napravio toliko popularnim da je postao jezik za ugradnju (engl. embeddable language) opšte namjene.

Konkretno, Mozilla implementationSpiderMonkey je ugrađen u nekoliko okruženjima, kao što su Yahoo! Widget Engine. Ostali programi sa ugrađivanjem ECMAScript uključuju Adobe proizvode Adobe Flash (ActionScript) i Adobe Acrobat (za skriptnim PDF datoteke).

Tcl (engl. Tool Command Language) je nastao kao proširenje jezika, ali se počeo više koristiti kao jezik opšte namjene u ulogama sličnim jezicima Python, Perl i Ruby. S druge strane, REXX izvorno nastao kao jezik kontrole poslova (engl. job control language) posao kontrole, ali se često koristi i kao proširenje jezika i kao jezik ošte namjene

Tcl karakteristike

- Sve operacije su naredbe, uključujući i jezičke strukture. Oni su pisani u prefiks notaciji.
- Naredbe obično imaju operatore ili funkcije sa promjenljivim brojem argumenta (engl, variadic)
- Sve može biti dinamički redefinisano i preklopljeno (engl. overloaded)
- Svi tipovi podataka se mogu manipulisati kao stringovi uključujući i izvorni kod
- Interfejsi upravljani događajima (engl. event-driven) utičnice (engl. socket) i datoteke. Upravljanje vremenom (engl. time-based) i upravljanje bazirano na korisničke događaje su takođe mogući.
- Vidljivost varijabli je ograničena na statički opseg
- Sve Tcl naredbe same generišu poruke greškama i pogrešnoj upotrebi
- Proširivost, preko C, C + +, Java i Tcl.
- Interpretirani jezik pomoću bytecod emetoda
- Za različite platforme (engl. cross-platform): Windows API, Unix, Linux, Macintosh, itd.
- Integracija sa prozorima (GUI) interfejsa TK.
- Postoje više mehanizama distribucije
- BSD licence, slobodno distribuirane

Ostali složeni i zadatku orijentisani programi mogu uključiti ugrađene jezike za programiranje i tako dopustiti svojim korisnicima više kontrole i dati im veću funkcionalnost nego može biti dostupna kroz korisnički interfejs, bez obzira koliko oni bili sofisticirani. Na primjer, Autodesk Maya 3D autorski alati koristi ugrđeni jezik MEL skriptni jezik, Blender koristi Python za ovu ulogu.

Neke druge vrste aplikacija koje trebaju veću brzinu (na pr. mehanizam igrica) takođe koristi ugrađene jezik. U tou razvoja, to im omogućava da brže razviju prototip bez potrebe da korisnika poznaje unutrašnju strukturu aplikacije. Skriptni jezici koji se koriste za ove namjene su u rasponu od poznatijih Lua i Python do manje poznatih poput AngelScript i Squirrel.

Ch je još jedan C kompatibilan skripting opcija za industriju za ugradnju u C/C++ aplikacione programe.

L1: Uvod u skripting jezike

Zaključak*
Nakon studiranja sadržaja ovog poglavlja, studenti će steći početna znanja iz oblasti skripting jezika

LearningObject

Uvod u skripting jezike

Uvod

Skripting jezici su danas znatno napredniji od svojih daljih predaka, po pitanju performansi, sintakse i mogućnosti samih jezika. Originalno zamišljeni da prošire funkcionalnost programa pisanih u drugim jezicima (uglavnom statički tipiziranih i kompajliranih), ili za pisanje kratkih skripti u shell okruženjima (na primer bash, awk, csh), skripting jezici su evoluirali u ozbiljne jezike generalne namene. Samim tim, veliki broj jezika koje danas nazivamo skripting jezicima su u isto vreme i generalni 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.

- Deklaracija jezika kao „skripting jezik“ više nema važnost koju je nekada imala upravo iz ovih razloga, ali 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)

Interaktivno okruženje (REPL)

Većina skripting jezika (svi na ovom predmetu) imaju interaktivno radno okruženje, nazvano **REPL** (read-eval-print-loop). Ovo okruženje omogućava korisniku da pokrene interpreter za određeni skripting jezik, i u njega unosi komande liniju po liniju (ili nekoliko linija od jednom). Kod koji korisnik unese se odmah izvršava, a greške ili rezultati koda su vidljivi odmah nakon njihovog unošenja. Ovo olakšava istraživanje sintakse, dostupnih modula i funkcija, ali i praktičan rad u nekim slučajevima – na primer programer koji radi sa nekom bazom podataka pomoću određene ORM ili wrapper biblioteke, može da pokrene REPL, uključi odgovarajuće module, i interaktivno isprobava i testira ideje nad bazom podataka.

Interaktivno okruženje za Ruby

Pokretanjem komande *irb* (**I**nteractive **R**u**b**y Shell), dobija se interaktivno okruženje za Ruby jezik. Posle pokretanja, irb tipično ispiše verziju interpretera i jezika i prompt:

1.9.3-p448 :001 >

Odavde je moguće kucati ruby kod direktno u interaktivni interpreter, koji će automatski ispisivati povratne vrednosti izraza koji se izvršavaju:

1.9.3-p448 :001 > 5 + 2 * 1.4

=> 7.8

1.9.3-p448 :002 >

Irb čuva istoriju unesenog koda kojoj se može pristupiti strelicama gore i dole. Takođe pruža auto-complete za poznate objekte, pa je moguće napisati "hello".swapc<TAB> da se dobije "hello".swapcase.

Interaktivno okruženje za Python

Interaktivno okruženje za Python se dobija jednostavnim kucanjem komande python u shell. Python će pritom ispisati verziju interpretera i još nekoliko informacija.

Python 2.7.4 (default, Apr 19 2013, 18:28:01)

[GCC 4.7.3] on linux2

Type "help", "copyright", "credits" or "license" for more information.

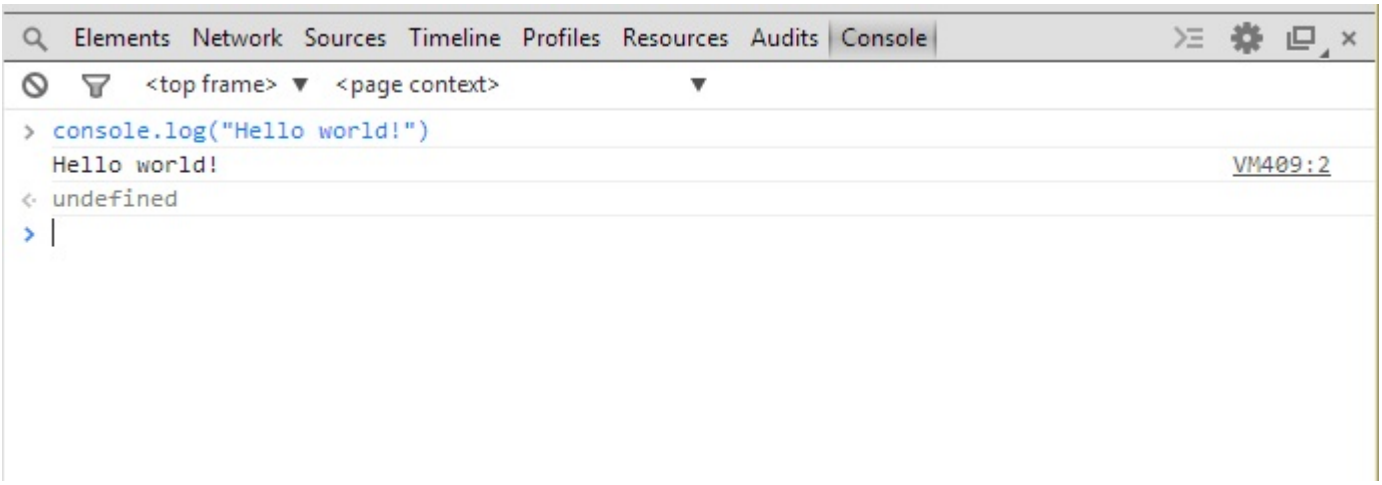
>>> print "Hello world"

Hello world

Kao i irb, i python REPL podržava istoriju i autocomplete.

Interaktivno okruženje za JavaScript

JavaScript je jezik koji radi na nekoliko različitih platformi, pa zavisno od toga gde se koristi, interaktivni interpreter se pokreće na drugačije načine. Ukoliko se koristi JS u okviru nekog pretraživača, potrebno je otvoriti development konzolu, prečicom Ctrl-Shift-J (prečica za Google Chrome).



Slika1 Chrome development konzola

Ukoliko se koristi Node.js, moguće je koristiti node REPL za direktno izvršavanje javascript koda:

$ node

> console.log("Hello!");

Hello!

undefined

Undefined u ovom primeru predstavlja povratnu vrednost funkcije console.log.

Interaktivno okruženje za Perl

Perlovom REPL-u se može pristupiti uz pomoć komande perl sa parametrima:

- -d – koji govori perlu da se pokrene u debug modu
- -e 1 – koji govori interpreteru da evaluira svaki korisnički input

```
$ perl -d -e 1

Loading DB routines from perl5db.pl version 1.33

Editor support available.

Enter h or `h h' for help, or `man perldebug' for more help.

main::(-e:1): 1

DB<1> print "Hello World\n"

Hello World
```

Smernice za pisanje koda

Svaki od ova četiri jezika ima definisane smernice koje opisuju kako bi kod trebalo da izgleda.

Generalne preporuke

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.

Uvod u Perl jezik

Programski jezik Haskell

Zadatak je da analizirate i napišete izvještaj o funkcionalnom programskom jeziku Haskel.

Haskell je funkcionalna programski jezik. opšte namjene s jakim tipiziranjem (engl. strong static typingI). Objasniti šta to znači.

Vaš izvještaj treba da uključi slijedeće elemente;

- Uvod u Haskel
- Haskel variable i tipovi
- Haskel nizovi (engl. array) i liste
- Haskel funkcije
- Haskel Input/Output
- Rad sa datotekama
- Haskel klase i preklapanje (engl. overloading)

Napisati Haskel program da objasnite navedene karakteristike.

Zadaću poslati na adresu: nemanja.stojanovic.799@metropolitan.ac.rs

Naslov mejla treba da bude:

IT2008-IPT-SCRP-STRL-DZ-DZ01-Ime-Prezime-BrojIndeksa.docx