25.02.07
סקריפטים

המרצה: אליוט יפה – דוקטורנט למדעי המחשב
הקורס נמשך חצי סמסטר ובמהלכו יינתנו שלושה תרגילים – כל אחד לכשבועיים. התרגילים כמראה יהוו 60% מהציון הסופי.
הבדיקות של התרגילים הן אוטומטיות ולכן חייבים לעשות בדיוק מה שרשום וערעורים בטח לא יתקבלו.

☺ ☺ ☺ ☺ ☺

גם בשפות תכנות יש תיאוריה!

<u>שפות פרוצדורליות ושפות פונקציונליות</u>
http://en.wikipedia.org/wiki/Procedural_programming
http://en.wikipedia.org/wiki/Functional_programming
אנחנו מכירים בעיקר שפות פרוצדורליות. זה אומר שיש מכונה במחשב ועוברים ממצב למצב. יש במעבד program counter והוא עובר על הפקודות ומריץ אותן אחת אחת לפי הסדר. יש משתנים שזה מקום בזיכרון שאנחנו אומרים לו מה הוא. למשל, c, java, perl
בשפות פונקציונליות זה קצת שונה. אין להן סדר בהוראות. הן מחפשות אין לעשות את מה שמבקשים מהן. אין להן מצבים ואנחנו אפילו לא יודעים באיזה מצב מתחילה התכנית. למשל, Haskell, ML, prolog

<u>שפות אימפרטיביות ושפות דקלרטיביות</u>
http://en.wikipedia.org/wiki/Imperative_programming
http://en.wikipedia.org/wiki/Imperative_programming
בתכניות אימפרטיביות סדרת הפעולות היא זאת שמוגדרת והמטרה אינה מוגדרת באופן ישיר. בתכניות דקלרטיביות המטרה היא זאתצ שמוגדרתת במפורש ודווקא הדרך להגיע אליה היא מוגדרת בעקיפין.

<u>סוגים של משתנים</u>
http://en.wikipedia.org/wiki/Type_system
ב-static typing חייבים להגדיר את סוג המשתנה והוא אינו יכול לשנות את הסוג במהלך התוכנית. כלומר הוא יכול לקבל רק ערכים מסוג מסויים אחד שהוגדר עבורו.
ב-dynamic typing כל משתנה יכול לקבל כל סוג של ערך (מספר, רשימה, מחרוזת, פונקציה וכו'). לא מגדירים את סוג המשתנה מראש אלא במהלך הריצה. כמובן זה מוריד את קריאות הקוד ולכן חשוב לתת שמות בעלי משמעות ולתעד היטב.

<u>סוגים חזקים וסוגים חלשים</u>
http://en.wikipedia.org/wiki/Weak_typing
http://en.wikipedia.org/wiki/Strongly-typed_programming_language
בשפות שהם weakly-typed משתנים יכולים להחליף את הסוג שלהן במבלך הריצה. למשל ב-VB הקוד הבא הוא חוקי:

```
var x = 5
var y = "hi"
x + y            // 5hi
```

לעומת זאת, בשפות שהן strongly-typed ברגע שהכנסנו ערך למשתנה הוא קובע את הסוג שלו עד סוף הריצה. ב-python הקוד הבא אינו חוקי:

```
v = '1.2'
x = 5 * v        // runtime error
```

שפות בטוחות ושפות לא בטוחות

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

שפטת בטוחות (כמו java) לא מאפשרות לכתוב קוד שיכול להפיל את התכנית. לעומת זאתף בשפות לא בטחות (כמו c) יש מגוון רחב של אופציות לגרום לתכניות לעוף. אז למהלכתוב ב-c? – כי היא מאפשרת לכתוב תכניות שהן embedded ברמה נמוכה למעבד.

שפות נומינטיביות ושפות סטרוקטורליות

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

בשפה נומינטיבית מגדירים את סוג הפרמטרים של הפונקציה ואת סוג ערך ההחזרה שלה.
בשפות סטורקטורליות לא מגדירים את הסוגים ואז השפה מחפשת ובודקת מה אפשר לעשות.
בשפות duck השפה יודעת מראש מה סוג המשתנים והפעולות שניתן לבצע עליהם. למשל ב- python לכל אובייקט יש יש רשימה של פעולות אז אם יש בקוד a+b השפה הולכת ל-a ושואלת אותו אם יש לו פעולת +.

ניהול זיכרון

בשפות שהן explicit מבקשים שטח זיכרון ויכולים לעשות איתו מה שרוצים ובסוף צריך לשחרר.
בשפות שהן implicit לא צריך ללהגדיר טח. זה נעשה אוטומטית ע"י השפה. הבעיה היא שיכול לצאת מזה בזבוז די גדול של זיכרון כשהשפה כל הזמן תקצה עוד ועוד זיכרון בעוד אם היינו מנהלים אותו בעצמינו אולי היינו ממחזירים זיכרון. אמנם, החיסרון הגדול בשפות explicit הוא שיש הרבה יותר מקום לטעויות של המתכנת.

במצגת יש טבלה שמרכזת שפות ואת התכונות שלהן.

http://en.wikipedia.org/wiki/Perl
http://en.wikipedia.org/wiki/Python
http://en.wikipedia.org/wiki/Ruby_%28programming_language%29
http://en.wikipedia.org/wiki/Ruby_on_Rails
http://en.wikipedia.org/wiki/Model-view-controller

Perl פותחה ע"י איש מחשבים שרצה לשלב תכונות של כמה שפות שהוא עבד איתן – shell, AWK, SED ו-c. Perl היא שפה ישנה יותר מ-python. אפשר לעשות מה ממש כמעט הכל ויש יותר מדרך אחת לעשות כל דבר (TIMTOWTDI) שזה נחמד. אבל השפה קצת מלוכלכת. אז המציאו את python שהיא יפה וגם בה אפשר לעשות הכל אבל היא קריאה יותר ויש דרך אחת לעשות את הדברים. חוץ מזה היא גם מונחית עצמים.
Ruby היא ערבוב של perl ו-python. היא יותר מונחית עצמים מ-python אבל יש בה את הלכלוך של perl. לא משתמשים כמעט ב-ruby. מה שמשתמשים בו זה ruby on rails שזו מסגרת שפותחה מ-ruby והיא משתמשת לבניית אתרי אינטרנט. העבודה מחולקת לשלוש מסגרות שיש להן ספריות משלהן – model, view, controller (MVC). ה-model מגדיר את האובייקטים שיש באתר והקשרים ביניהם. ה-view מתאר איך הדף נראה וה-controller אחראי על ה-business logic. מה שטוב בחלוקה הזאת הוא שהיא מאפשרת לטפל בכל מישור בנפרד. בדיקות הן חלק מ-ruby on rails שבונה אוטומטית את המסגרת של הבדיקות. זאת מסגרת טובה לכתוב אתר במהירות ובצורה מסודרת. הבעיה עם זה היא שאי אפשר לכתוב אתרים גדולים שיפעלו במהירות כין אין שם threads אז אי אפשר לעשות דברים במקביל.

בשפות סקריפט אין מהדר. הקוד ישר מורץ במחשב ללא שלבי ביניים. יש שרואים בזה יתרון גדול ויש שרואים בזה חיסרון גדול.

שיטת העבודה המקובלת היא מפל. בהתחלה רושמים מסמך ארוך של דרישות, אז מחליטים מה המשאבים שדרושים, אז עושים design ואז ממשים את הכל ובסוף בודקים ועושים את התיקונים הדרושים עד שהמוצר יכול לצאת לשוק. כאן הכל מתבצע אחד אחרי השני ואין סוגי עבודה שונים במקביל. שיטת עבודה אחרת היא פיתוח תוכנה זריר (agile). בשיטת עבודה זו יש דגש על עבודה בצוות

ועושים את מה שצריך כשאפשר ולא מחכים לכל מיני מסמכים. אין הגדרת דרישות רשמית אלא מתארים איך המערכת צריכה להיראות מהצד של המשתמש וכותבים בדיקות בשביל לבדוק אם זה הגיוני בכלל.
אין design מסודר (מה שיכול כמובן ליצור בעיות) ולכל יחידה שכותבים עושים בדיקות בנפרד.
כשמסיימים מריצים את הבדיקות של המערכת השלמה. השיטה הזאת היא כמובן מאוד בעייתית למנהלים משום שהיא לא מסודרת ואין לו"ז ברור. אין כמעט חברות שעובדות ככה. בד"כ עובדים בשיטת המפל ומוסיפים לה דברים מהפיתוח הזריז. למשל, כל מנהל שמכבד את עצמו יבדוק את כל היחידות בנפרד ולא יחכה לסוף כדי לגלות ששום דבר לא עובד...

במצגת יש תוכניות פשוטות ב-perl וב-python.

ש"ב – לכתוב תכניות פשוטות ולהריץ אותן

11/03/07
סמינר

מי מבין הרבה יותר ממך?
ה אוטומטים הראשונים ובונים על ה אופציות מאקמק.
תוכנית בדיקת תשפ את כל אחת. שומרים את כל
ה מצבים הראשונים ובונים על ה אופציות מאקמק.
ענית שרוצים לקפוץ את abab|abbb.
ומשם ראשית מכונה כזאת:

כך זה תוכנן. אבל יש שתי בעיות להסתכל על זה.
ראשית כשמתחילים אכראות לאן עליית יד. שנית קלים
ואז תוצרים אחורה ומנסים אם יש אופציה אחת.
תוכנית בדיקת תשפ את כל אחת. שומרים את כל
אוטומטים ראשונים. מה זה שנקרא
מיטויים ראלריים והם הולכים לשנות את החיים שלנו.
יש אוסף של אותיות ורוצים להבין מישהו אחד. למשל
רוצים למצוא אם יש אומרים רפלים של the או למשל
אנחנו היינו עושים את זה בראווה? היינו עושים שלמה שעומדת
אות אות ומפקידת מצבים אחרים. לא כך!
מיטויים ראלריים אלגוריתמים מחשוביות - בראמינים סופיים
... תוכנניים. ראמינים יודעים לקפץ שפות.
ab - שרשור של שתי אותיות
a|b - a או b
a+ - a אחד או יותר
a? - a יום או אחד.
מאחיות יש ביטויים של אכונות שאקפלות את הדברים האלה
ענית שרוצים לקפץ את abab|abbb.
וכשרוצים לשרטט אכונה כזאת:

הנה עוד השפה:

- . - מקום כל אות. למשל .t מתאים כל דבר שמתחיל ב-t ואחריו יש אות אחת כלשהי.
- $ - סוף של שורה
- ^ - תחילת שורה
- \b - תחילת מילה או סוף מילה (אפס אותיות שהן לא whitespace)
- \B - כל דבר שאינו תחילת מילה. למשל /\Bbab/ מתאים לשלוש הראשונות ב- babab
- * - 0 או יותר אותיות
- + - אות אחת או יותר
- ? - 0 או 1 אותיות
- [] - אחת מהאותיות שבתוכם מתאימים. אפשר להשתמש ב- "-" בתוך זה כדי לציין טווח של אותיות. [a-d] = [abcd]
ו- ^ אחרי שהכוונה היא לכל תפ מהאותיות פרט ל- [^a-o].
- סוגריים עגולים משמשים ואז צריך להתייחס כאל אות אחת. למשל /Th(eatr)e?/
ויתן להתאים גם ל- Theatre וגם ל- The.
- \w - אינדוס ח פחמן ומקסימום m פחמנים
\w+ - מינים m פחמן
- \d - a או c, והוא שווה ערך לאות שלפניו.
למשל /t(h|r)e/ מתאים ל- the או tre
ואילו /th|re/ מתאים ל th או ל- re.

Regex ב- Perl:

```
"abcde" =~ m/a.*e/
```

m - מוצא התאמה. match ומחזיר אותו או שקר.

```
$string = "crop"
$string =~ s/cr/st/..
```

s - מחליף את המופע הראשון של cr ב- st.
אם היינו מוסיפים את /g/ זה היה מחליף את כל המופעים.

(5)

```
$string = "A number is 12345";
$string =~ /.*(\d)+/;
```

ה- * מקרה את כל המספר כולו, אולם הוא משים לוב את הוא כל אחד אחרונה (\d) והיה מפך "5".
כבה התאפיינות אחסון. כלומר, הוא שקט כל אחד שנוצר ורק הוא כל אחד אחרונה אחד אחרונה.

מאפיינת ש כ אינו פונקציה - תכונות, אחזקות, ואחר ואחר.

שם - התחבר לאתר www.cs.huji.ac.il/~sclass

- שני תלקים - 1) בוחנים מעבדה - פרטים
2) שקפים בוחנים מעבדה.

קופץ ראשון:

login
כל אחד
כל אחד
כל אחד
כל אחד
כל אחד
כל שנה בוחנים מעבדה

כל אחד אחרונה ה- כל אחד אחרונה

קופץ שני:

login
כן
no, —, —, —
אחד אחרונה אחד אחרונה
אחד אחרונה אחד אחרונה

⑥ 18.3.07
סקריפטים

התרגיל: להגישה עד חצות
מטלה: יש למלא את יכולה להיות קבלת באישור הבדיקה
שמות הבאו: חד פעמי השיעור בשעה לך ביתה 4

PERL

ב perl יש המון פרטים ועשות מינו דברים, יש אולם
ענק של ספריות. כמו יתרון וגם חיסרון.

לעומת הוא איש system שאפשר באיזו אוניברסיטה והיו לו
לא מיני בעיות. הוא יודע שלא ינקסס עם שפות סקריפטים
שהיו את. אולם הישו לנו אהב את לה את הטו המלצו שפה.

למשתנים ב perl אין סוג כלשהו. לא צריך להגדיר אם לה
תהי או char או string. וגם אותו דבר.

$myVar = 'c';
$myVar = "dina";
$myVar = 23;

סימן של סקלר

יש שני סוגים של מחרוזות: אם המחרוזת בין גרשיים לא
כפולים, למשל 'this is $myVar' את לה קורה כמו
שהיה רשום וכתוב. אם כה מרשים כפולים "this is $myVar"
את הטו פותח את המחרוזת ומחפש משתנים, את מוצא
אם $myVar='cat' את ולכן "this is cat".

השפה היא weakly typed. יש בה באמת רק שלושה:
Scalar, arrays of scalars, hashes.
המשתנים מצוינים ע"י האות הראשונה- sigil.

סקלר הוא משתנה יחיד - פשוט איזה ערך. הוא מסומן ב- $.
סקלר שעוד לא אותחל יש ערך undef. אל תשכח
לבדוק אם למשתנה כזה יש ערך - defined($myVar)
וזה מחזיר true אם הוא מכיל ערך ו- false אחרת.
נשים ❤ שלא צריך להסביר על משתנה, לא צריך להיפרד את הסוג
אלא/צריך רק להקצות ליפון. כל ניתנו הליפון נעשה מהופן
אוטומטי ע"י perl.

רשימה - מסומנת ע"י @ ומכילה בתוכה סקלרים בשם.
הרשימה מתחילה באינדקס 0.

@myVar = (3, "foo", 'c')

כדי לגשת לאחד, למשל שאנחנו מפנים, הוא סקלר משתנים
ב- $: $myVar[1] == "foo"

אפשר להכניס לתוך רשימה שלישיה שונים. למשל
אפשר להגדיר @foo[1,3] = ("bear", "dear") ואז במקום
0 ובמקום 2 יהיה undef.
אפשר לשרשר רשימות: @foo = (@foo, "elk")
@foo = ("ace", @foo)

איך יודעים מה אורך של רשימה:
אפשר להשתמש ברשימה // $size = scalar(@foo);
ואינדקס הכי גדול ברשימה // $size = $#foo;
מ 0 ועד $#foo = scalar(@foo) - 1.

אפשר להשתמש ברשימות עוצמתיות שמעל לאתחל משתנים:
($first, $last) = ("John", "Moreland")
ואז $first הוא John ו- $last הוא Moreland.
הדבר הכי הוא תמהון. אל אם רושמים ($a, @b, $c) = (1,2,3,4,5,6)
יקרה ש- $a = 1, @b = (2,3,4,5,6) ו- $c הוא undef.

בנוסף ישנה קיצורי דרך. אפשר להשתמש ב- .. כדי להשתמש בסדרות תווים.

(1..5) = (1, 2, 3, 4, 5)

(1.3 .. 6.1) = (1.3, 2.3, 3.3, 4.3, 5.3)

("a" .. "z") = ("a", "b", "c", ..., "z")

והכי מגניב: ("aa" .. "ab") = ("aa", "ab", ..., "zz")

מאפיין יש רשימה של פונקציות שאפשר לעשות על השימות.

hashes - מסומנים ע"י % - משתנה מקבילות של אסוציאטיבי עם ערכים. בלי סקלרי יכול להכיל אסוציאטיבי.

% days = ('M', "Monday", 'T', "Tuesday");

ונקבל טבלה כזאת:

key	value
M	Monday
T	Tuesday

ואם נוסיף $days{'W'} = "Wednesday" אז נקבל מוסף שורה שנכה.

מאפיין יש רשימה של פונקציות שאפשר לעשות על hashes.

פונקציות -

```
sub MyFunction {
    //code
    return $value; //optional
}
```

אם הפונקציה מוגדרת אחרי שמשתמשים בה נכון לקרוא לה עם MyFunction&. אחרת לא נכון כאשר ה- & נלמדת פרמטרים: באופן מוכן לא מגדירים פרמטרים. לפונקציה מקבלת רשימה של פרמטרים שנמצאים ב- @_:

```
sub MyFunction {
    ($arg1, $arg2, @list) = @_;
    $arg1 = $_[0];
}
```

פרמטרים עוברים בשיטת by value כלומר אנו אומרים לתוכנית.

בהגדרת a חפץ היא שלך והמשתנים הם שלמספרים. אפילו אנחנו
לא הגדרנו במקום שהקצינו בתוך לולאות, לכן משנה אותה.
אפשר להגדיר משתנים לוקליים בתוך הפונקציה - my $arg1 = 1.

אפשר לתת לתוכנית ארגומנטים בהרצה.

$0 = השם של התכנית

$argv[0] = הארגומנט הראשון

$argv[1] = הארגומנט השני

⋮

ב Perl יש את כל הפקודות שיש בכל השפות -
תנאים, לולאות, IO, קבצים.
פורום אפשר מאוחר.

Advanced PERL

איך כותבים קוד קריא ויפה?

- use strict - אוסף אותנו להכריח על משתנים שלמים לפני השימוש
- use warnings - אומר מידים עם -w - את כל האזהרות שלא עזרו שלנו
אין, בדרים שעושים דברים לא עשויים כי אפשר אפשר עזרו שלנו

- כותבים את הפונקציה עם הפרמטרים של מי זה?
אם יש רשימה של הפרמטרים האחרונים לא יקראו ריקים:
Myfunction ($scalar1, @list, $scalar2)
↓
לא יבואו ריק

- פוינטרים - reference על סוג יודעים בשפה אי הוספת \
לפני זה - dereference עצמו הפוך מקלף.
כשבנו עושים dereference מוסיפים לפני את הסוג
של הסוג הכלוני.

- יש ע אין אפשרות שמופיעים אינו דברים עד קבלים - למשל אם קופץ
קיים, אם הוא קריא, אם כן תיקיה, אם לא וכו' - כל כך יהיה
שימושי לתכנון.

- eval - ליות שרוצים לבדוק משהו שהמשתמש נתן מתוך פרמטר - למשל
לחלק. regex זה צריך להמיר אותו מסתם אחרוכרת לפקודה שעושה
- map - כי אפשר ליצור hash

- grep - עם אפשר בשימוש ולהפעיל אצה פונקציה. למשל אפשר ליצר
מערך השורות שמתאימות ל regex מסוים.