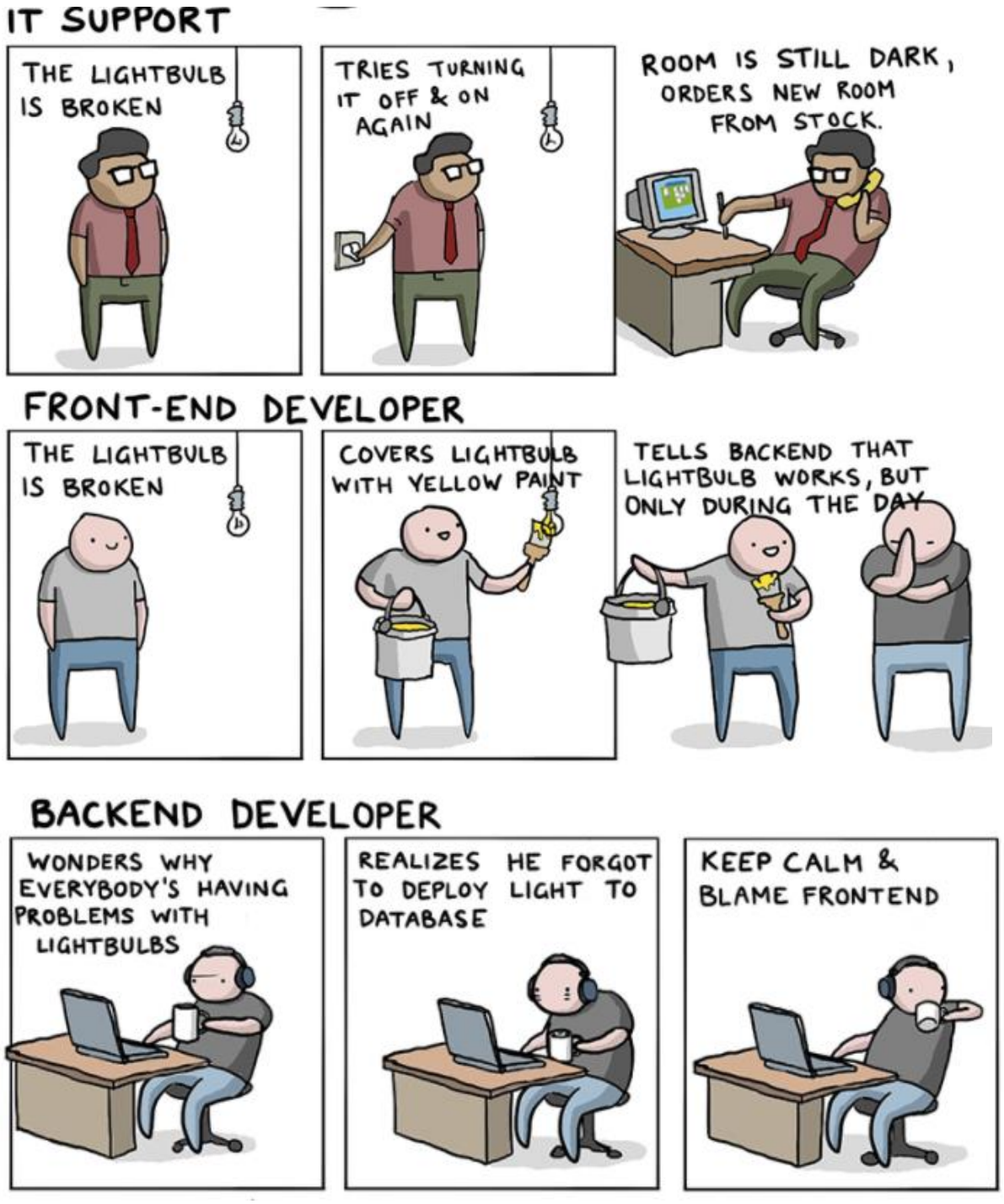
אלון קווארט 201025228 kwart@campus.technion.ac.il
יונתן בתן 302279138 yonibettan@gmail.com



1. קראו על בקרת זרימה (Control Flow) בשפת Kotlin. מהו הדמיון ומהו השוני בין if-else של Kotlin לבין if-else של שפת C? מהו הדמיון ומהו השוני ביחס ל if-then-else של ML? האם לאופרטור הטרינארי בשפת C קיים מקביל בשפת Kotlin ? אם כן מהו ? אם לא, מדוע? בתשובתכם התייחסו גם לערכם של ביטויים ולטיפוסיות ערכים אלו.

בשפת C if-else הוא statement **שאינו מחזיר** expression, אלא קובע האם statement או קבוצה של הצהרות תתבצע, כלומר קובע תנאי לסיעוף.

מנגד ב- Kotlin - if-else הוא statement **היכול להחזיר** expression. בסגנון כתיבה הדומה לשפת C המתכנת בוחר להתעלם מערך ההחזרה של ההצהרה, אך אם הוא בוחר להחזיר ערך הוא מתחייב להחזיר ערך במידה וההתניה אינה מתקיימת. בדוגמה הבאה ניתן לבצע מספר הצהרות בתוך הסקופים אשר כל סקופ מחזיר ערך אחר. כלומר היא מתנהגת כפונקציה.

```
val max = if (a > b) {
        print("Choose a")
        a
    } else {
        print("Choose b")
        b
    }
```

בשפת `Kotlin` אין את האופרטור הטרינארי אך מאחר ו`if` מחזיר ערך ניתן לבצע אותו בצורה הבאה אשר מקבילה לה:

```
val max = if (a > b) a else b
```

בשפת `ML` – `if-then-else` הוא `statement` אשר **מחזיר** expression **בלבד**, כלומר מקיים רק את הדוגמה השנייה לעיל.

2. האם פונקציה היא טיפוס בשפה? הסבירו. האם פונקציה ב Kotlin יכולה להיות פולימורפית ? אם כן, מה ההבדל בין פונקציה שכזו ב Kotlin לבין פונקציה פולימורפית בשפת ML? ב Kotlin ניתן להגדיר פונקציה ב infix notation, השוו בין הגדרה שכזו בשפת ML לבין זו שב Kotlin (ההשוואה צריכה לכלול : מתי ניתן להגדיר, ואיך ההגדרה מתבצעת)

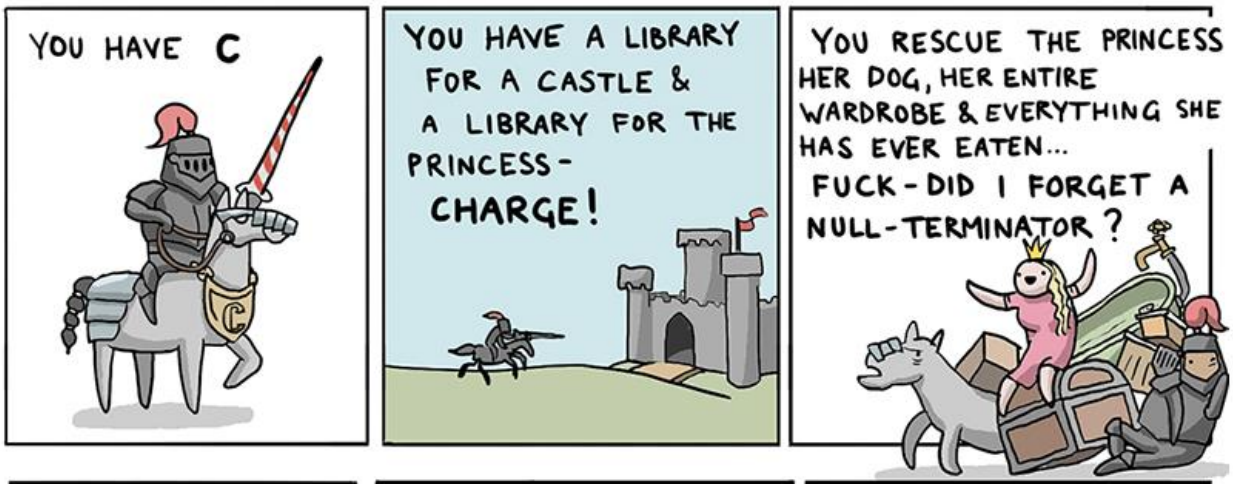
- מאחר ויש ב Kotlin בנאים לפונקציות, נסיק כי פונקציה היא טיפוס (את ההנחה שיש בנאי לפונקציות קיבלנו מהערה של גיא על תרגיל בית 2).
- פונקציות יכולות להיות פולימורפיות, ניתן לראות זאת בכך שניתן להגדיר מספר פונקציות בעלות אותו השם אשר יקבלו פרמטרים שונים ויחזירו ערכים שונים.
  - ב-ML ניתן להגדיר פונקציה אשר תקבל ארגומנט כללי a', תוכן הפונקציה הנקראת אינו משתנה כתלות בטיפוס הפרמטר.
  - ב-Kotlin פונקציה תיקרא על בסיס סוג הארגומנט המועבר, ולכן תוכן השגרה יכולה להיות שונה לחלטין.
- infix notation – בשפת ML ניתן להכריז על כל מילה, החוקית כפונקציה, כ-infix , ללא תלות בסוג הארגומנטים המועברים לפונקציה אך מספר הארגומנטים צריך להיות 2.
  בשפת Kotlin ניתן להגדיר infix בתנאים הבאים:
  - הפונקציה שייכת למחלקה או הוספה למחלקה (מאחר וניתן להוסיף פונקציות בזמן ריצה)
  - הפונקציה מקבלת ארגומנט יחיד.
  - הפונקציה מסומנת על ידי המילה השמורה infix במקום הכרזתה.

3. הסתכלו בשקפים של פרק 3. מהי תכונת Void Safety? האם Kotlin היא Void Safe? אם כן, הסבירו כיצד תכונה זו באה לידי ביטוי בשפה. אם לא, תנו דוגמת קוד שמוכיחה זאת. אל תפספסו את האופרטור של Elvis.

בשפה ערכים אשר יכולים לקבל ערך ריק, NULL, צריכים להיות מוגדרים כך בעת אתחולם באמצעות אופרטור '?' המסמל לבנאי כי ערך זה עלול לקבל ערך ריק בעתיד: `var b: String? = "abc"; b = null`

אך בד בבד השפה עלולה לקבל ערך ריק משפה חיצונית ולכן אינה בטוחה לחלוטין. השפה מגדירה מספר דרכים לבדוק ערכים כנ"ל, לדוגמה:

- האופרטור של אלביס: `val l = b?.length ?: -1`
  ? הוא אופרטור המבצע בדיקה האם הערך b ריק, במידה ואינו ריק מתבצעת ההצהרה הראשונה, אחרת ההצהרה לאחר סימן ה- :?.
- אופרטור !! : `val l = b!!.length`
  במידה ו-b ריק, נזרקת שגיאה בתוכנית.



4. קראו את הסיכום על פולימורפיזם. מהם סוגי הפולימורפיזם הקיימים בשפת Kotlin, הסבירו בפירוט והביאו דוגמאות לכל אחד מן הסוגים (לפחות דוגמה אחת לכל סוג)
   - Ad-Hoc –
     - Coercion- מתבצעות המרות סמויות, לדוגמה בתרגיל בית הקודם עיקר הבאגים היו מחיבור מספרים בעלי טיפוס שונה.
     - Overloading- קיימת בשפה:

```
fun printNumber(n : Number){
    println("Using printNumber(n : Number)")
    println(n.toString() + "\n")
}

fun printNumber(n : Int){
    println("Using printNumber(n : Int)")
    println(n.toString() + "\n")
}

fun printNumber(n : Double){
    println("Using printNumber(n : Double)")
    println(n.toString() + "\n")
}
```

   - Universal-
     - Parametic- בדומה לשפת C++ ניתן להגדיר פונקציות ומחלקות גנריות אשר יפעלו על מספר לא מוגבל של טיפוסים, אך בדומה ל C++ צריך להגדיר אותם בזמן קומפילציה.
       לדוגמה:

```
fun <T, R> List<T>.map(transform: (T) -> R): List<R> {
    val result = arrayListOf<R>()
    for (item in this)
        result.add(transform(item))
    return result
}
```

- Subtyping – נתמך בשפה, בדוגמה הבאה ניתן לראות כי הפונקציה something מצפה לטיפוס Animal ובפועל מקבל טיפוס Dog היורש מAnimal.

```
/* Alon Kwart 201025228 kwart@campus.technion.ac.il
   Yonathan Bettan 302279138 yonibettan@gmail.com  */
open class Animal {
    open fun make_noise(){
        return
    }
}
class Dog : Animal() {
    override fun make_noise() {
        println("Wof Wof")
    }
}
fun somthing(animal: Animal) {
    println("pls compile")
    animal.make_noise()
}
fun main(args : Array<String>) {
    val puppy = Dog()
    somthing(puppy)
}
```

```
Run Ex3Kt
"C:\Program Files\Java\jdk1.7.0_80\bin\java" ...
pls compile
Wof Wof

Process finished with exit code 0
```

5. הגדירו מהו type punning (או תרגמו את השקף המתאים). תנו דוגמאות ל type punning בשתי השפות : C, Rust.

Type punning – היכולת לנתח ייצוג מכונה (קידוד בינארי\ביטים) של ערך באופן שאינו תואם או עקבי לו.

C:

```
void foo() {
    int* p = 1;
}
```

בקוד לעיל, המצביע מאותחל לכתובת 1, אך כתובת זאת אינה כתובת חוקית עבור מצביע מסוג int מאחר והכתובות הנמוכות הינן כתובות שמורות. מאחר וזאת פקודה לא טובה שהמתכנת יכול לעקוף, ראינו בה Type Punning.

RUST:

```
let bitpattern = unsafe {
   std::mem::transmute::<f32, u32>(1.0)
};
assert_eq!(bitpattern, 0x3F800000);
```

std::mem::transmute ממיר ערך מייצוג אחד לייצוג השני באותם הביטים, כלומר הוא לוקח את הביטים המקוריים ומניב את ערכם על פי פירוש הביטים עם הטיפוס החדש, שלב זה עובר Type punning בהמרה. מעבר לכך, שפת RUST היא strongly typed בעוד בכדי לבצע Type punning על השפה להיות שפה בעלת טיפוסיות חלשה.

6. כתבו תכנית קצרה בשפת C המדפיסה LE אם ערכים מיוצגים בזיכרון ב Little Endian, ומדפיסה BE אם ערכים מיוצגים בזיכרון ב Big Endian. הסבירו בקצרה מדוע התכנית מבצעת את הנדרש.

```
#include <stdio.h>

int main () {
    unsigned int x = 0x76543210;
    char *c = (char*) &x;

    if (*c == 0x10) {
        printf ("LE");
    } else {
        printf ("BE");
    }
    return 0;
}
```

גודל המילה של X הוא 32 ביטים, כלומר 4 בתים. סידור הבתים בשתי התצורות הוא בסדר הפוך, כלומר במחשב בו הערכים שמורים ב LE הערך שיתקבל אחרי ההמרה הוא הבית התחתון, כלומר 0x10 מנגד, ב BE בו הבתים שמורים בסדר הפוך הערך שיתקבל לאחר ההמרה יהיה 0x76.

7. הסבירו מדוע Union בשפת C לא מממש בצורה מדויקת את בנאי הטיפוסיDisjoint Union

בשפת C ה-Union משמש לשמירת מספר סוגים של משתנים באותה הכתובת בזכרון. בדומה לכך ה- Disjoint Union מאחד למשתנה אחד את היכולת להשתמש במספר בנאים. ההבדל המהותי בין השנים הוא כי בשפת C לא ניתן לקבוע מה מהו הטיפוס השמור בunion על בסיס המטא דאטא (תמיד אפשר לקרא כל ערך ולהניח מה שמור, אבל לא שמור ערך המייצג את הבנאי ששמר שם מידע).

8. הגדירו מהם Mixed typing ו Gradual typing. מהו ההבדל בין המושגים?
   - Mixed Typing:
   מגדיר כי השפה תומכת הן בטיפוסיות סטטית והן טיפוסיות דינאמית. כלומר נעשות בדיקות טיפוסים הן בזמן ריצה והן בזמן קומפילציה.
   - Gradual Typing:
   מגדיר כי השפה מבצעת בדיקות טיפוסיות בצורה דינאמית, כלומר בזמן ריצה, אך לעיתים ניתן להגדיר משתנים שהטיפוסיות שלהם תיבדק בזמן מוקדם יותר כגון קומפילציה.

9. למדו על סוגי המערכים מהשקפים בפרק 5.2 .באיזו שיטה של ייצוג מערכים משתמשת שפת התכנות NIMROD? הביאו ציטוטים התומכים בתשובתכם.
בשפת NIMROD המערכים הם מסוג סטטי, כלומר גודל ותוכן המערך נקבעים בזמן הקומפילציה. עם זאת, ניתן באמצעות קוד קצר לשנות את מפתחות המערך ולכן יש לו נטייה להיות אסוציאטיבי.

"The arrays in Nim are like classic C arrays, their size is specified at compile-time and cannot be given or changed at runtime.
The size of the array is encoded in its type and cannot be accidentally lost.
Therefore, a procedure taking an array of variable length must encode the length in its type parameters.
Alternate methods of indexing arrays are also allowed, the first type parameter is actually a range (just a value, as above, is syntactic sugar for `0..N-1`). It's also possible to use ordinal values to index an array, effectively creating a lookup table"

https://nim-by-example.github.io/arrays/

YOU HAVE PASCAL
YOU DECLARE YOUR PRINCESS, CASTLE & RESCUE PLAN
THEN YOU GO FOR A DRINK & FORGET ABOUT THE IMPLEMENTATION