**Programozási nyelvek**

Minden programozás ilyen

Két kérdésre ad választ

* Mit kell megoldani
* Hogyan kell megoldani

Két nagy család:

* Deklaratív
* Imperatív (C#, Java,…)

Deklaratív (**Mit?**)  
- Elég deklarálni a problémát és keretrendszer találja ki az algoritmust  
- Funkcionális prog nyelvek (Lisp → Scheme → **Racket**) – Nincs ciklus, van rekurzió  
- Meta Language (ML) – SML (Standard ML)

Imperatív (**Hogyan?**)  
- Egy feladatot bárhogy megoldhatok (A programozó találja ki az algoritmust)

Logikai programozási nyelvek:  
- Prolog (nincs visszaadott érték)

Procedurális  
- Fortran (fizikusok használják leginkább)  
- Cobol (régi)  
- PL/1  
- Modula80  
- Pascal  
- Ada  
- Delfi  
- C (Kernighan & Ritchie)

MOV AX,0

XOR AX,AX – kinullázni register

Szivárgó absztrakció elve (Leaky abstraction)  
<https://www.joelonsoftware.com/2002/11/11/the-law-of-leaky-abstractions/>

**HF: Két bekezdésnyi jegyzet erről ↑**

Procedurális prog nyelvek:

* DRY (*Do not repeat yourself*)
* Lehet metódust írni (alprogram[eljárás, függvény])
* Robert C. Martin – nem magadnak programozol, hanem más programozónak, ezért tiszta kódot kell írni (clean code könyv)

Kötelező olvasmány lista:

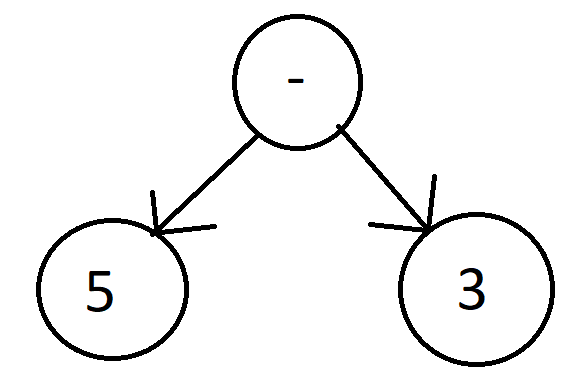
* Robert C. Martin – Tiszta kód
* Robert C. Martin – The clean coder (Túlélőkönyv programozóknak)
* Principles, Patterns, Practices (PPP)
  + Best Practices
  + Design Patterns

Best practices:  
- Scrum → XP (Extrém programozás)

* Ha valamit csinálok, akkor ezt extrém mód csinálom, ha nem, akkor extrém módon nem
* Innen jönnek a best practice-ek
* Code review (átnézés, inspekció) – Valaki más kódját nézem át, bemutatom a kódomat
* Amit minden cég használ, az best practice
* **Teszt vezérelt programozás** (Test driven development - TDD)
  + Sokan szeretik, kevesen használják
* **Páros programázás** (Pair programming – Driver-Copilot)
  + Cégen belül így könnyű átadni a konvenciókat
  + Minél hamarabb találunk meg hibát, annál olcsóbb a kijavítás
  + Ha az egyikük elmegy, akkor a másik ismerni fogja a működést

**2. óra:**

**Bináris fa:**

Egy bináris fát 3 féle képpen lehet bejárni: Inorder, Preorder, Postorder

Inorder: 5 – 3

Preorder: - 5 3

Postorder: 5 3 –

Racket

Feladat mindhárom nyelven: Fibonacci n-edik elemének meghatározása

Feladat2: Java, C#: Embereket tartalmazó lista (public String név, public double fizetés,public string telefonszám).

Írassuk ki a legnagyobb fizetésű ember (linq vagy stream).

Mindenkinek növeljük a fizetését 10%-al. (C# select, Java map).

Keressük azokat a telefonszámú embereket, akiknek 70-es a száma. (0670…)

**3. óra**  
**Osztály jellemzői:**  
- Felülete (interface)  
- Megvalósítása (implementation)

**Objektum jellemzői:**  
- Típus (type):  
 - Lényegében ugyanaz mint az interface  
- Viselkedés (behavior):   
 - Futó kód  
 - Függ a belső és a külső állapottól  
 **class Kutya{  
 boolean mérges;  
 public string ugat(){  
 if (mérges) {  
 return “rrrrr”;  
 }  
 else {  
 return “vau vau”;  
 }  
 }  
}**

- Belső állapot (inner state):  
 - Mezők pillanatnyi értéke  
  
- Külső állapot:   
 - Többi objektumnak a belső állapota  
 - Egy részét megkaphatom paraméterként és lényegében a paraméterektől függök  
  
- Stateless osztály:   
 - Egy mező sincs!  
 - Ha csak egy vagy egy állapotom sem lehet  
   
Immutable:   
 - megváltozhatatlan (nincs setter, a mezők sosem szerepelnek értékadás bal oldalán, csak   
 construktorban)  
  
Side effect:  
 - Megváltoztatod a környezetedet (pl. képernyőre írás)  
 - Saját mezőnek értéket adni nem side effect, csak másnak  
  
Globális változó:  
 - Nincs szükség paraméterátadásra, rövid, gyors  
 - Veszélyes, mert elronthat más részeket, megtalálhatatlan hibákat eredményezhet  
 - A mező osztályon belül globális változó  
  
**Deklaratív oldalon** (funkcionális programozásban) nincs side effect, vagy nehezen használható