Egy tábor margójára

Aszalós László[1], Bakó Mária[2]

[1]aszalos.laszlo@inf.unideb.hu, [2]bakom@unideb.hu
Debreceni Egyetem, IK és GTK

Absztrakt. Az informatika iránti érdeklődés felkeltésének egyik eszköze a nyári informatikai tábor. A Debreceni Egyetem Informatika Kara egy pályázat[1] révén jutott olyan lehetőséghez, melylyel ilyen táborokat tudott szervezni. Az első nyáron csak kezdők részére hirdetett tábort, de többen olyanok is érdeklődtek, akik már túl voltak az alapozáson. Épp ezért a pályázat második nyarán már mind a kezdő, mind a haladó programozói tábor elindult. Ebben a cikkben ez utóbbit kívánjuk bemutatni, a feldolgozni kívánt tematikát, a felhasznált eszközöket, illetve a tapasztalatainkat.

Kulcsszavak: tehetséggondozás, funkcionális paradigma, Racket programozási nyelv

1. Tervezési fázis

Az informatika mára igen szerteágazóvá vált. A nyolcvanas években a BASIC volt az egyetlen széles körben elérhető programozási nyelv (a maga sokféle nyelvjárásával), így minden kezdő ezzel ismerkedett meg. A sokak számára elérhető *home computer* is ezt a programozási nyelvet ismerte alapból, ezt lehetett használni pár másodperccel annak bekapcsolása után. Csak a haladó felhasználók érezték meg ennek a nyelvnek a korlátait, és fordultak más programozási nyelvek iránt, melyek között hazánkban a Forth és az assembly igen előkelő helyet szerzett meg. A későbbiekben a Basic helyét átvette a Pascal, majd ezt követően már nagyon hosszúvá vált az elsőként tanítandó programozási nyelvek listája. Szinte vallásháború van a mai napig az egyes nyelvek evangélistái között, senki nem kíván engedni a saját kedvencéből. Ennek eredményeképpen a haladó táborunkra jelentkezettek előismereteinél előfordult a C#, C++, Python, JavaScript, sőt a LabView is.

Ha a tábor programozási nyelvének ezek egyikét választottuk volna, akkor az a pár személy – aki már jól ismeri ezt a nyelvet – unatkozott volna, amíg az összes többit sikerül az ő szintjére felhozni. Persze egy akár több éves előnyt nem lehet egy hét alatt behozni, így mindenképpen szétszakad a társaság, amit felettébb megnehezíti a tábor megszervezését és lebonyolítását úgy, hogy minden résztvevő azt érezze a végén, hogy hasznosan töltötte az időt.

Egy huszárvágással elintéztük, hogy minden táborozó azonos szinten legyen, ez pedig a kezdő szint volt. Habár az előbb felsorolt nyelvek mindegyikében elérhető az objektum-orientált paradigma, jellemzően ez nem kap szerepet az általános iskolai, illetve középiskolai oktatásban. (Most tekintsünk el a szakirányú szakgimnáziumi képzéstől.) Azaz a hagyományos, imperatív megközelítés az általános – még ha az adott nyelv más paradigmákat is támogat –, azaz az imperatív megközelítésben szocializálódott minden résztvevő. Ebben a szellemben fogalmazzák meg az egyes problémákat megoldó algoritmusokat. Annak érdekében – hogy a komfortzónájukból kimozgassuk őket –, egy teljesen más paradigmával kívántuk megismertetni a diákokat, ez pedig a funkcionális programozás volt.

Fontosnak tartottuk, hogy más módon, más szemszögből lássák az esetleg korábban már megismert feladatokat, kérdéseket. Érzésünk szerint nem volt elrugaszkodott célunk, csupán egy kis betekintést kívántunk adni egy számukra addig ismeretlen területre. Az már az adott diák érdeklődésén,

[1] EFOP-3.4.4-16-2017-00023 AZ MTMI szakokra való bekerülést elősegítő innovatív programok megvalósítása a Debreceni Egyetem vonzáskörzetében

kíváncsiságán múlik, hogy a tábor egy hete után szeretne még ezzel a paradigmával foglalkozni – akár ilyen irányú programozási nyelve(ke)t használva – vagy az általa eddig használt nyelvben fedezi fel és veszi birtokba az ott található funkcionális eszközkészletet, vagy csak egy rossz élményként tekint majd vissza a táborra. Véletlenül sem kívántunk térítőként viselkedni – amit többször is hangsúlyoztunk számukra a tábor folyamán –, ám meg akartuk mutatni, hogy a programozásban is vannak különféle irányzatok.

Az egyetemi oktatásban nyert tapasztalatok révén már tudjuk, hogy a hallgatók készek bármilyen nyakatekert szerkezetek használni, csak hogy ne kelljen eltávolodni a már elsajátított tudástól. Ezért készek megerőszakolni a programozási nyelveket, sokkal hosszabb, gyakran követhetetlen megoldásokkal előállni, mintsem átvegyék az adott rendszer filozófiáját és rövid, lényegre törő programot készíteni. A legeklatánsabb példákkal Prolog nyelven találkoztunk, ahol nincs sem hagyományos értelemben vett ciklus, sem feltételes utasítás. Ugyanez a hozzáállás – az új elsajátításának elutasítása – buktatta meg a kilencvenes években az Oberon nyelvet [1], melynek az a volt nagy bűne, hogy nem szerepelt benne `for`-ciklus, csak `while`.

Habár egyes szerzők szerint szinte bármilyen programnyelven lehet a funkcionális paradigmát követve programozni [2] [3] [4] [5] [6]; olyan nyelvet kívántam választani, melynek már a tervezésekor is ez a paradigma állt a középpontjában. Ilyen nyelv bőséggel található, a Lisp-familia igencsak tekintélyes. A Scheme elnevezésű variánst sikerült annyira leegyszerűsíteni, hogy több könyv is született azzal a céllal, hogy az olvasója elkészítse a saját Scheme variánsát [7] [8] [9], másrészt évtizedekig ezen tanulták meg a programozás alapjait a legnevesebb amerikai és európai egyetemek diákjai. A programozási nyelv és a hozzá kapcsolódó programozási környezet kiválasztásánál arra is kellett ügyelni, hogy itt középiskolás és még fiatalabb diákok fogják használni a programozói környezetet, akik nem biztos, hogy minden hibaüzenetet tökéletesen megértenek angolul, illetve nem feltétlenül a parancssor megszállottjai. Mindezeket mérlegelve esett a választásunk a Racket programozási nyelvre [10], és annak DrRacket programozási környezetére. Ez a nyelv is egy Scheme variáns, korábbi változatai még DrScheme néven futottak. Viszont pont oktatási szempontok miatt tértek el annyira a Scheme nyelvtől, hogy már illendő volt más elnevezést választani helyette.

Ha valaki utánanéz ennek a nyelvnek, akkor az azzal hirdeti magát, hogy programozható programozási nyelv, azaz valójában egy keretrendszernek tekinthető, melyben mindenki elkészítheti a saját programnyelvét [11]. A rendszer telepítésekor már több tucat ilyen nyelv megjelenik a számítógépünkön, kezdve prezentációkat, dokumentációkat leíró nyelvektől (slideshow, scribble) különböző bonyolultságú, kezdőknek szánt programozási nyelvekig (htdp, sicp, regex). Egy új programozási nyelv, vagy akár csak egy DSL megalkotása is önmagában megtöltene egy nyári tábort, de ahhoz már nagyon haladó diákokra lenne szükség. Ezért erről az irányzatról, erről a lehetőségről nagyon mélyen hallgattunk. A tábor során csupán a funkcionális programozás alapjait kívántuk megmutatni, semmi többet.

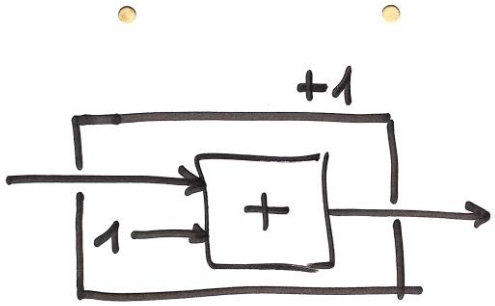
Szerencsére a Racket könnyedén telepíthető, így az egyik géptermünket bő egy óra alatt elő tudtuk készíteni a táborra.

Miután a táborban 20 gyerekkel foglalkoztunk, és tíztől tizennyolc évig terjedt az életkoruk, ezekből az adatokból nem lehet statisztikailag megalapozott következtetéseket levonni, így csak tendenciákat tudunk megfogalmazni a – nem mindenki által kitöltött – kérdőívünk, valamint személyes tapasztalataink során. Miután ezekkel a diákokkal csak a tábor idejére találkozunk, és legközelebb majd csak a következő táborban, alapvetően szemléletmódot, egy más nézőpontot kívántunk átadni nekik.

2. Elméleti alapozás

Több olyan tantárgyat tartottunk már egyetemisták számára, ahol egy teljesen új, esetleg eltérő paradigmájú nyelvvel kellett megismertetni őket. Miután ebben az esetben sincs királyi út, elegendő időt kell számukra hagyni, hogy akklimatizálódjanak az új paradigmához, átálljon a gondolkodásuk. Ezt a

periódust mi *agymosásnak* nevezzük, mert meg kell szabadulni a régi berögződésektől, hogy el lehessen sajátítani az új ismereteket, az új látásmódot.



1. ábra: Az eggyel növelés függvényének rajza

A [12] könyvhöz hasonlóan a funkcionális paradigma tanításakor az első géptermi gyakorlatok valójában táblás gyakorlatok, mert a gépeket nem kapcsoljuk be. *Táblán programozunk*, azaz oda rajzoljuk fel a nagyon egyszerű programjainkat. Például az összeadás egyik argumentumát fixálva elkészülhet az 1. ábrán szereplő *inc* függvény. A rajzolt függvényekkel találkozás minden egyes életkorban meglepi a diákokat, de elég gyorsan bemelegednek, és onnantól kezdve igen gyorsan tudunk haladni. Pár példa után már önállóan is elkészülnek a *programok*, és ekkor térhetünk át azok *futtatására*. A táborban kezdetben aritmetikai függvényeket kellett megfogalmazni, majd mikor ezek mindenki számára már jól mentek, ezután megismerkedtünk a lista adatszerkezettel. Természetesen ezt a tudást is az adatszerkezetekhez kapcsolódó függvényekkel mélyítettük el. Miután a lista egy rekurzív adatszerkezet, így az ezt feldolgozó függvényeknek is rekurzívaknak kell lenniük. Nyilvánvalóan ez a diákok többségének nehézséget jelentett, mert korábban a rekurzióval nem igazán találkoztak. Rendszerint ciklusok segítségével oldották meg mindazt, amire itt alapesetben rekurziót kell használunk. Arról, hogy különféle kiegészítések révén a Scheme nyelveknél is találhatóak ciklushoz hasonló szerkezetek – pontosabban lista-értelmezések – mélyen hallgattunk, mert az egyik célkitűzésünk a rekurzió megismertetése, és használatának begyakorlása volt. Ennek megfelelően nagyon sok feladatot, és annak megoldását mutattuk be, melyeknél rekurzióval igen könnyen megfogalmazható a megoldás.

További újdonságot jelentett a Lisp-típusú nyelveknél elterjedt prefix jelölés, és az a rengeteg zárójel. Miután a függvényszimbólumok és operátorok a megelőzik az argumentumaikat (prefix jelölés), a megszokott `1+2-t` `(+ 1 2)` formában kell írni. Ez a szokatlan sorrend sok panasznak volt forrása, sok időre volt szükség az átálláshoz. Viszont annyi ideig nem tartott a tábor, hogy a diákok értékelni tudják ennek a sorrendnek előnyös vonásait, például amikor elemzőt kellene írni a forrásnyelvi program feldolgozásához. A megoldások bemutatása *live coding* formájában ment, így nem egy kész programot töltöttünk be az IDE-be, és magyaráztuk el sorról-sorra, hogy mit is látnak maguk előtt; hanem ott született meg a megoldás előttük, esetleg több lépésben. A Lisp-típusú nyelvekre jellemző módon a Racket is rendelkezik egy REPL elnevezésű parancssorral, ahol zenész módjára lehet improvizálni: rövidebb hosszabb kifejezéseket kipróbálni, amit a rendszer azonnal ki is értékel. Így be tudtuk mutatni, hogy bizonyos kifejezések hogyan viselkednek, és ezekből – esetleg több lépésben – állítottuk össze a megoldást.

A vicc szerint a Lisp a *lots of irritating superflous parentheses* mozaikszava. Egy-egy függvénydefiníció végén valóban nem ritka a tucatnyi egymás mellett álló záró zárójel. De ha összehasonlítanánk egy-egy méretes C/C#/Java programot, hogy abban hány (kerek, szögletes és kapcsos) zárójel van, és a neki megfelelő Lisp/Scheme programban hány zárójel szerepel, meglepetéssel vennénk tudomásul, hogy az utóbbiban van kevesebb. Egy kis gyakorlat után, esetleg erre kihegyezett szövegszerkesztő használata esetén (paredit, parinfer, rainbow-brackets) már egyáltalán nem zavaróak a zárójelek, sőt

az újabb Scheme verziók – a Racket-hez hasonlóan – megengedik a szögletes és kerek zárójelek keverését, így még nyomtatott formában is jól olvashatóak a programok. Persze egy hét a megszokáshoz kevés, a diákoknak nem vált ennyi idő alatt a kedvencévé a Łukasiewicz féle prefix jelölésrendszer. A többszintű listák ellaposításának programja így néz ki nyomtatásban, míg egy arra alkalmas szövegszerkesztőben a különböző szinten szereplő zárójelek automatikusan más színűek lesznek, illetve amikor mozgunk, folyamatosan jelzi a szövegszerkesztő a megfelelő zárójelpárt.

```
(define (flatten lista)
  (cond
    [(empty? lista) '()]
    [(list? (first lista))
     (append (p07 (first lista)) (p07 (rest lista)))]
    [else  (cons (first lista) (p07 (rest lista)))]))
```

A prefix jelölésrendszer megismerése, és egy kis jártasság megszerzése után a felrajzolt definíciókat elkezdtük Racket nyelven is megfogalmazni. Az előkészítésnek hála, ez nem okozott gondot senkinek sem. Ezzel véget is ért az első nap. A második nap a legalapvetőbb programozási szerkezeteket ismertük meg, hogy hosszabb, bonyolultabb programokat írhassunk: `if`, `cond`, `lambda`, `let`, `let*`, `letrec`, `map`, `filter` és `foldl`. Nem vettük ezek mindegyikét használatba, de bemutattuk őket, mert ez volt az, amire építkezni kívántunk a tábor során.

Miután a szó elszáll, az írás megmarad, az elhangzott elméleti ismeretek már a tábor kezdete előtt elérhetőek voltak a tábor Moodle kurzusában, így ha valakinek valami volt teljesen érthető, vagy hiányzás miatt nem hallotta, utólag elolvashatta azt. Sőt, aki többre vágyott, az további kapcsolódó anyagokra (honlapokra, könyvekre) mutató hivatkozásokat talált a Moodle kurzusban.

3. Gyakorlati feladatok

Miután a tábor programozói tábor volt, minden napra, minden témához egy-egy feladatsor társult, melyet a Moodle kurzusban, magyar nyelven tettünk elérhetővé. Az elméleti alapozás során a bevezető Scheme könyvek anyagából válogattunk, mint pl. [13]. Majd a *99 Prolog feladat* Lisp-re átírt változatából [14] szemezgettünk, és fogalmaztuk át a feladatokat és megoldásaikat Racket-re. Ebben a feladatsorban a listakezelési és aritmetikai feladatok voltak a központban, innen származik az előbb bemutatott program is.

Ezután komolyabb vizekre eveztünk, a Euler Projekttel [15] ismertettük meg a diákokat. Itt többnyire aritmetikai, számelméleti feladatok vannak százszámra, és regisztráció után be lehet küldeni a saját megoldásunk által generált eredményt (mint egy számot). A rendszer nyilvántartja, hogy mely feladatokat sikerült megoldanunk, ami hatalmas ösztönzés volt a diákjaink számára. Amint elkészültek egy önálló megoldással, vagy elkészültünk egy közösen elkészített megoldással, azonnal töltötték is fel az eredményeket, és boldogan fogadták a jó választ jelző hatalmas zöld pipát. Az Euler Projekt különféle szintű feladatot tartalmaz, és gyakran nem mindegy, hogy milyen programnyelven próbáljuk megoldani azt. Volt egy feladat, ahol 100 darab ötvenjegyű szám összegének a kezdete (első tíz számjegye) volt a kérdés. Ez a méret már nagyobb annál, hogy egy hagyományos egész típusú változóban elférjen. Tehát a megoldáshoz rendszerint implementálni kell egy megfelelő adatszerkezetet, valamint ezen az adatszerkezeten az összeadást. Nem így nálunk. A parancssorba át kellett másolni a számokat, egy zárójelpárt kellett írni köréjük, majd a nyitó zárójel után be kellett szúrni egy + jelet. Ezzel el is készültünk, a rendszer már adta is vissza a megoldást. Az jó kérdés, hogy azzal, hogy megadtunk egy kifejezést, valójában programoztunk-e, vagy sem; de a kívánt eredményt minimális erőbefektetéssel megkaptuk. Hasonló meglepetést okozott a diákok számára a 1000 faktoriális kiszámítása, ami több

sorból álló számot adott eredményül, szinte egy szempillantás alatt. Az ehhez szükséges program az alábbi, amely veszi *n*-ig a számokat, melyekből elhagyja a kezdő nullát, majd a megmaradt számokat mind összeszorozza. Természetesen használható a megszokott rekurzív összefüggés is, amellyel majd dupla ilyen hosszú lesz a megoldás. A Racket képességeinek hála nekünk nem kell foglalkozni a túlcsordulással.

```
(define (f n) (apply * (rest (range (add1 n)))))
```

Talán ez a pár példa néhány diáknak felnyitotta a szemét, és tudatosította, hogy nem biztos, hogy elegendő csak egy programnyelvet ismerni, és minden egyes feladatot azon megoldani, mert létezhetnek olyan rendszerek, ahol egyes dolgok egyszerűbben, könnyebben fogalmazhatóak meg.

A tábor kezdetén alapvetően mi oldottuk meg a feladatot, folyamatosan magyarázva, hogy mit miért is csinálunk, időnként táblára rajzolt szerkezetek segítségével. Ahogy haladtunk előre az időben, egyre gyakrabban már a diákok mutatták meg saját megoldásaikat, melyeket átbeszéltünk, hogyan lehetne esetleg még javítani rajta.

A tábor utolsó napján – egyfajta kitekintésként – a 2048 játék [16] egy Racket nyelvű megoldását mutattuk be lépésről lépésre, rámutatva, hogy a grafikus felület kezelése a programkód szinte elenyésző részét képezi. A program túlnyomó részét a játék szabályainak megfogalmazása, az adatszerkezetek definiálása, a hozzá kapcsolódó függvények elkészítése tette ki. A tábor utolsó óráiban – kifogyva az előkészített feladatokból – túlléptünk a korábban megszabott korlátokon és egy prímszámokhoz kapcsolódó feladatot próbáltunk megoldani. Ebben az esetben fontos volt a korábban kiszámolt prímszámokhoz való visszatérés, azaz az azokat tároló adatszerkezet. Miután a lista elérése lineáris, hosszú listák esetén ez jelentős lassulást okoz. Ezért megmutattuk a asszociatív tömbök (hash) használatát, ami már kicsit körülményesebb, mint az addig látottak; de nem kívántuk azt a hamis látszatot kelteni, hogy ebben a nyelvben minden egyszerűen megoldható.

4. Vélemények, tapasztalatok

Fontos körülmény, hogy a tábor teljes mértékben ingyenes volt a diákok számára. Emiatt néhány diák lemorzsolódott a hét során. Kicsit más lett volna a helyzet, ha jelentős összeget kellett volna fizetni a szülőknek, mert akkor valószínűleg futottak volna a pénzük után, és nem engedik, hogy a gyerekük ellógjon egy napot, vagy napokat. Viszont több mint kilencven százaléka a diákoknak kitartott, és néhányuk csak az iskolai évnyitó miatt fejezte be az utolsó napot korábban. A többiek még péntek délután négykor is ott verték a billentyűzetet, és gépelték a Racket nyelvű programokat. Úgy véljük, hogy ez a legnagyobb dicséret.

A feladatokat próbáltuk úgy megfogalmazni, hogy egymásra épüljenek, időnként visszatértünk egy korábbi megoldáshoz, és azt kellett továbbfejleszteni, hogy megoldjunk egy későbbi feladatot. Viszont volt olyan diák, melynek az új feladat kitűzésekor már kész volt a megoldása, mert a korábbi feladat megoldása során nem elégedett meg a minimális megoldással, próbálta továbbgondolni, új feladatot tűzött ki saját maga részére, és meg is oldotta. A tábor Benjaminja hasonló módon elkalandozott, és megszállottan próbált egy általa kitalált – az általunk megoldott feladatoktól teljesen eltérő – feladatot megoldani. Az egyik szünetben – pár speciális programozási szerkezettel megismertetve – sikerült begyújtani a rakétáit, és még megszállottabban haladt a saját feladata megoldása felé.

A *generációs* különbségek természetesen itt is kiütköztek. Az általános iskolások jellemzően szívesen begépelték a kivetítőn megjelenő teljes megoldásokat, de ha csak hiányos megoldással találkoztak, akkor vártak volna a sült galambra. Ilyenkor mindenkihez külön-külön leülve, átbeszélve a megoldandó feladatot rendszerint már képesek voltak kitölteni a hiányzó részeket. Meglepő módon esetükben a hét túlnyomó részében sikerült megküzdeni a mobil és a netes játékok csábításával, csak a napok

végén, illetve az utolsó napon – mikorra szellemileg már teljesen elfáradtak – választották a kikapcsolódásnak ezt a formáját. A középiskolások viszont a tempónkat jól bírták, gyakran még egymással is versenyeztek, hogy ki milyen gyorsan tud megoldani egyes feladatokat, illetve érdeklődéssel nézték egymás programjait, hogy kinek milyen eszközt használva sikerült megoldani a feladatot.

Miután ez volt az első ilyen táborunk, a jövő nyári tábor sikeressége céljából elkészítettünk egy kérdőívet, melyben anonim módon megfogalmazhatták a véleményüket a táborozók. Sajnos kevesebb, mint a diákok fele válaszolt, éppen a kemény mag. Ők elégedettek voltak a táborral, tetszett nekik a választott programozási nyelv, szívesen jönnének jövőre is; és úgy néz ki, hogy sikerült megfertőzni őket, mert egy megszokott és széles körben ismert programnyelvbeli tudásuk elmélyítése helyett egy újabb, a fősodortól távoli nyelvet szeretnének megismerni. Ezzel rendesen feladták a leckét számunkra, mert valami újabb, különc paradigmát kellene keresni számukra. Hiányosságként a szünetek száma és terjedelme lett megfogalmazva, amit meg kell szívlelnünk; valamint az étkezést érte még panasz (nem a minőséget, nem a mennyiséget, hanem inkább a szerkezetét), amin szintén könnyen lehet javítani.

5. Összefoglalás

Először szerveztünk most nyáron haladó programozói tábort a Debreceni Egyetemen. Emiatt volt bennünk egy kis bizonytalanság, mert ismeretlen terepre tévedtünk. Ezt tetéztük azzal, hogy a szervezés megkönnyítése érdekében egy, a diákok számára ismeretlen nyelvet választottunk, melyben azért mi sem voltunk teljesen otthonosak. Mindenesetre a felkészülés során igen sok feladatot lefordítottunk magyarra és meg is oldottuk, majd megosztottuk a feladat kiírását és a megoldását a tábor Moodle kurzusában. Ezzel sikerült olyan gyakorlatot szerezni, mellyel a tábor megtartása zökkenőmentes volt.

Mint a diákok tábor során nyújtott teljesítménye, mint a kitöltött kérdőívek alapján sikeresnek tekinthető a tábor, úgy érezzük sikerült egy új világot kinyitni előttük. Ezzel a saját magunk számára kitűzött célunkat elértük. Ennek megfelelően a soron következő tábort ebben a szellemben kívánjuk megszervezni jövőre, s már csak egy kérdésre kell választ találni addig, hogy mi legyen az az egzotikus nyelv, mellyel jövőre megtámadjuk a diákokat?

Irodalom

1. M. Reiser és N. Wirth: *Programming in Oberon: Steps Beyond Pascal and Modula*, Addison-Wesley (1992)
2. P.-Y. Saumont: *Functional Programming in Java: How functional techniques improve your Java programs*, Manning Publications (2017)
3. I. Cukic: *Functional Programming in C++: How to improve your C++ programs using functional techniques*, Manning Publications (2018)
4. L. Sheehan: *Learning Functional Programming in Go: Change the way you approach your applications using functional programming in Go*, Packt Publishing (2017)
5. S. F. Lott: *Functional Python Programming: Discover the power of functional programming, generator functions, lazy evaluation, the built-in itertools library, and monads*, Packt Publishing (2018)
6. P. Hartel, H. Muller: *Functional C*, Addison-Wesley (1997)
7. C. Queinnec: *Lisp in Small Pieces*, Cambridge University Press (2003)
8. N. M. Holm: *Compiling Lambda Calculus*, Lulu Press (2018)
9. N. M. Holm: *Scheme 9 from Empty Space*, Lulu Press (2014)

10. PLT Research Group: *Racket*. (2019)
http://racket-lang.org. (utoljára megtekintve: 2019.9.15.)

11. M. Butterick: *Beautiful Racket*, szerzői kiadás (2019)
https://beautifulracket.com.

12. D. S. Touretzky: *Common LISP: A Gentle Introduction to Symbolic Computation*, Dover Publications (2013/1989).

13. B. Harvey, M. Wright: *Simply Scheme: Introducing Computer Science*, MIT Press (1999)

14. J. Meidanis: *L-99: Ninety-Nine Lisp Problems*, (2006)
https://www.ic.unicamp.br/~meidanis/courses/ mc336/2006s2/funcional/L-99_Ninety-Nine_Lisp_Problems.html (utoljára megtekintve: 2019.8.6.)

15. *Project Euler* (2019)
https://projecteuler.net/ (utoljára megtekintve: 2019.8.25.)

16. *2048*, (2019)
https://2048game.com/ (utoljára megtekintve: 2019.5.10.)