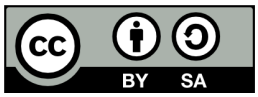
Didaktika programovania

doc. RNDr. Ľubomír Salanci, PhD.
Jihočeská univerzita v Českých Budějovicích, Pedagogická fakulta
© 2018

Tento vzdělávací materiál vznikl v rámci projektu
CZ.02.3.68/0.0/0.0/16_036/0005322 **Podpora rozvíjení informatického myšlení**.

EVROPSKÁ UNIE
Evropské strukturální a investiční fondy
Operační program Výzkum, vývoj a vzdělávání

Podléhá licenci Cretive commons Uveďte původ-Zachovejte licenci 4.0



Obsah

I. O tomto materiáli

Tento text je určený lektorom, vysokoškolským učiteľom, ktorí pripravujú budúcich učiteľov informatiky na vyučovanie programovania v rámci predmetu Didaktika programovania.

Pri písaní tohto dokumentu sme vychádzali z dlhoročných skúseností, ktoré sme postupne nadobúdali, keď sme najskôr učili programovanie a neskôr aj viedli predmety, kurzy alebo školenia zamerané na didaktiku programovania. Učili sme študentov, budúcich učiteľov informatiky, na našej fakulte. Boli medzi nimi aj veľmi slabí ale aj úžasne talentovaní programátori. Ukazovalo sa však, že didaktika bolo pre mnohých z nich, bez rozdielu na programátorské skúsenosti, nová, neprebádaná oblasť.

Pracovali sme aj s existujúcimi učiteľmi. Niektorí z nich chceli získať novú aprobáciu pre vyučovanie informatiky, takže s didaktikou programovania nemali žiadne skúsenosti, ale mohli čerpať zo svojich dlhoročných skúsenosti s vyučovaním iných predmetov a s práce s deťmi.
Viedli sme školenia aj pre aprobovaných učiteľov informatiky, ktorí mali bohaté skúsenosti s vyučovaním programovania na základnej alebo strednej škole. Mali však záujem o svoje ďalšie vzdelávanie a didaktika programovania im pomohla potvrdiť si svoj správny prístup, ale aj získať nové didaktické poznatky, napríklad pre vyučovanie programovania v nových programovacích jazykoch.

Snažíme sa používať nasledovnú označenie:

- žiak = žiak na základnej alebo strednej škole
- študent = študent na vysokej škole – budúci učiteľ
- lektor = učiteľ na vysokej škole, ktorý učí študentov

II. Rôzne didaktiky programovania

Kurzy didaktiky programovania môžu mať rôzne ciele, obsahy aj spôsoby, ako v nich lektori pracujú so študentmi. Tieto rôzne prístupy by sa dali rozčleniť do nasledujúcich kategórii:

- deformovaná didaktika programovania,
- jednoduchá didaktika programovania,
- súčasná didaktika programovania.

Zdá sa ná užitočné, aby sme o týchto variantoch vedeli. Môžeme sa tak vyhnúť nedorozumeniam, zbytočným komplikáciám, chybám alebo demotivovaniu našich študentov.

Deformovaná didaktika programovania

Namiesto didaktiky sa realizuje niečo iné, napríklad pokročilejší kurz programovania, riešenie náročnejších programátorských úloh (napríklad zo súťaží) alebo kurz všeobecnej pedagogiky.

Domnievame sa, že takýto stav je zapríčinený nasledujúcimi faktormi:

- Bol nedostatok poznatkov zo samotnej didaktiky.
- Nebolo jasné, čo sa očakáva od budúceho učiteľa informatiky.
- Študenti nemali samostatný predmet, na ktorom by sa naučili programovať, alebo mali tak slabé programátorské skúsenosti, že učiť ich didaktiku programovania nemalo zmysel.
- Lektori netušia, čo majú so študentmi robiť. Zdôvodňujú to tým, že študentov potrebujú *„nejako zamestnať“*, *„nejako užitočne s nimi stráviť čas“*.
- Stále existuje aj taká predstava, že *„stačí, ak bude učiteľ dobrý odborník – programátor, potom to bude automaticky znamenať, že je aj dobrý učiteľ informatiky“*.
- Existuje tiež presvedčenie, že študenti riešením náročných úloh získajú ďalšie námety pre šikovných žiakov, didaktika je vlastne zbytočná.

Nevýhodou takéto prístupu je, že študent sa nedozvie, ako má učiť bežných žiakov – teda začiatočníkov.

Jednoduchá didaktika programovania

Študenti sa učia navzájom nový programovací jazyk. Jeden študent vysvetľuje ostatným spolužiakom jednu tému, tí sa mu snažia porozumieť. Neskôr sa dostane k slovu ďalší študent, ktorý akoby vyučuje nasledujúcu tému atď. Tak sa postupne každý študent ocitne v úlohe učiteľa. Lektor do vysvetľovania nevstupuje, podnecuje však ostatných študentov, ktorí sú v úlohe žiakov, aby sa pýtali, ak niečomu nerozumejú.

Oproti predchádzajúcemu spôsobu vedenia didaktiky tu vidíme kvalitatívny posun:

- Náplň takejto didaktiky už súvisí so samotným vyučovaním – dôraz sa kladie na trénovanie vysvetľovania.
- Študenti prežívajú vyučovací proces – učia sa konštruktivisticky, teda sami zbierajú skúsenosti s vyučovaním svojich spolužiakov.
- Ostatní študenti hodnotia vyučovanie tým, že sa pýtajú svojho spolužiaka, komentujú jeho vystúpenie.

Nevýhodou takéhoto prístupu je, že vysokoškolskí študenti už poznajú mnohé programátorské pojmy a koncepty (vykonávanie príkazov, premenná, cyklus, vetvenie, pole atď.), nemusia ich už

svojim spolužiakom opätovne vysvetľovať. Rizikom potom je, že študenti sa nenaučia učiť začiatočníkov práve tieto základné a dôležité programátorské koncepty.

Súčasná didaktika programovania

Ako sme už na začiatku spomínali, pri formovaní súčasnej didaktiky programovania sme vychádzali:

- z vyučovania predmetu Didaktika programovania na našej fakulte,
- z našich praktických skúseností – čo sme potrebovali v prvom rade pre vlastné vyučovanie programovania na gymnáziu,
- zo skúseností s vyučovaním programovania niekoľkých tisícov začiatočníkov,
- z vedenia kurzov programovania aj didaktiky pre existujúcich učiteľov z praxe.

Chceli sme pri tom dôrazne rešpektovať cieľovú skupinu, ktorej adresujeme naše poznatky. Pre výskumníka by bolo pravdepodobne dôležité to, aby spoznal výskumné metódy, najnovšie vedecké výsledky, teórie, trendy alebo vízie.

Pre budúceho učiteľa informatiky sú dôležité predovšetkým tie poznatky z didaktiky programovania, ktoré sa bezprostredne týkajú jeho povolania, teda to, aby sa naučil správne postupovať pri vyučovaní. Zdá sa nám, že študenti najskôr ocenia praktickú časť didaktiky programovania – najmä návody a neskôr aj princípy toho, ako učiť jednotlivé témy alebo ukážky vhodných motivácií.

Ako dôležité sa nám ukázali **diskusie** so študentmi o tom, ako učiť žiakov riešiť problémy. Napríklad, ako žiakov postupne priviesť na myšlienku, že sčítanie čísel `0 + 2 + 3 + ... + 99` môžu v programe zapísať napríklad pomocou cyklu `for i in range(100)` a pripočítavania `s = s + i`. Chceme teda vychovať budúcich učiteľov tak, aby svojim žiakom nepredkladali hotové riešenia, ale aby ich, v prípade potreby, dokázali na správne riešenie naviesť.

Užitočné sú aj diskusie o vzdelávaní, cieľoch informatiky, učebniciach alebo maturitách, ale aj o forme vyučovania alebo o ťažkostiach, ktoré sú spojené s konkrétnymi témami.

Máme skúsenosti s tým, že takmer každú teoretickú informáciu navyše bude študent v tejto etape svojho poznania vnímať ako nepraktickú zbytočnosť. Často jej ani nebude rozumieť, lebo s vyučovaním v reálnej škole ešte nemá dostatok skúseností.

Naša súčasná predstava o náplni didaktiky programovania je takáto. Budúci učiteľ informatiky by mal:

- naučiť sa správne vyučovať základné programátorské koncepty,
- porozumieť a natrénovať si základné princípy vyučovania programovania, akými sú napríklad teória poznávacieho proces alebo zostavovanie nie slovníka,
- naučiť sa analyzovať programátorské témy a zostaviť si vyučovaciu hodinu,
- naučiť sa korektne hodnotiť svojich žiakov,
- poznať dostupné vzdelávacie materíaly,
- vedieť analyzovať a hodnotiť pedagogické dokumenty,
- vedieť hodnotiť programovacie jazyky z hľadiska ich vhodnosti pre vyučovanie programovania,
- poznať históriu vyučovania informatiky, súčasný vzdelávací systém, postavenie a ciele programovania na základnej a strednej škole.

Tieto kompetencie sa snažíme v čo najväčšej možnej miere rozvíjať formou aktivít tak, aby boli študenti do ich riešenia vždy aktívne zapojení. Ukázalo sa nám, že je potrebné, aby študenti získavali skúsenosti s vyučovaním tým, že prakticky riešia rôzne didaktické úlohy, problémy, vystupujú pred ostatnými spolužiakmi, analyzujú vystúpenia svojich spolužiakov a diskutujú o nich.

III. Cieľové skupiny

V tejto kapitole uvádzame naše praktické skúsenosti s rôznymi skupinami ľudí, s ktorými sme pracovali. Aj lektori zažijú rôzne situácie alebo reakcie svojich študentov. Niekedy nemáme dobrý návod, ako sa zachovať. Každopádne môžu o našich skúsenostiach rozmýšľať alebo sa na niektoré z ich dopredu pripraviť.

Počúvame rôzne, nie vždy pozitívne názory na to, ako ľudia vnímajú didaktiku a vyučovanie programovania.

Laická verejnosť

Toto je najširšia skupina ľudí. Sú to ľudia, ktorí nemajú pedagogické vzdelanie, neučia na základnej ani strednej škole. Školu a vyučovanie vnímajú z pohľadu svojich zážitkov, ktoré získali počas povinnej školskej dochádzky alebo ako rodičia, ktorých deti absolvujú povinnú školskú dochádzku.

Laická verejnosť didaktiku zväčša úplne ignoruje. Ešte častejšie však zisťujeme, že ľudia nepoznajú tento aspekt vzdelávania, netušia o tom, že existuje nejaká didaktika a že vyučovanie by sa tiež malo riadiť určitými pravidlami. Máme skúsenosti s tým, že samotné vyučovanie, nielen informatiky a programovania, verejnosť výrazne podceňuje. Nezriedka sa stretávame s názormi, že napríklad „*didaktika je zbytočná*“, „*stačí použiť zdravý rozum*“ a pod.

V diskusiách o vyučovaní programovania (napríklad aj na internetových fórach) sa nezriedka zvyknú vyjadrovať ľudia, ktorí sa javia ako veľmi zdatní programátori, a ktorí sa naučili programovať tým, že čítali knižky, manuály, svoje prvé programy zapisovali v textovom editore a komplikovali z príkazového riadka. Takýto diskutujúci potom často vyslovujú svoj názor: *„Ja som samouk. Keďže som sa týmto spôsobom naučil programovať a pritom som si to poriadne oddrel, tak toto je jediný správny spôsob, ako treba učiť.“* Sami pritom niekedy priznávajú, že nezažili dobré vyučovanie programovania. Z nášho pohľadu je takýto postoj egocentrický – diskutujúci sa vôbec nezamýšľajú nad tým, či by ich spôsob učenia sa vyhovoval aj ostatným žiakom. Domnievame sa, že takéto názory sú spôsobené malou, resp. žiadnou skúsenosťou s prácou s reálnymi žiakmi v triede. Navyše do svojich uzáverov časti nezahrnú ani fakt, že sami sa programovaniu venovali vo svojom voľnom čase veľa hodín, ale pre bežných žiakov je informatika iba 1 hodina do týždňa – jedna, možno nevýznamná, hodina z mnohých hodín v škole.

Iné názory počúvame od kolegov z akademickej obce, ktorí však nemajú skúsenosti s vyučovaním na základnej ani strednej škole, a nemajú ani skúsenosti s prácou s učiteľmi. Nemajú teda predstavu o problémoch, s ktorým sa učitelia stretávajú, ani o tom, akých prehreškov sa učitelia pri vyučovaní dopúšťajú. Domnievajú sa však, že učitelia nepotrebujú žiadnu didaktiku *„veď to nakoniec v praxi sami dáko zvládnu“*.

Odborníci, ktorí majú výborné matematické základy, sa niekedy o didaktike vyjadrujú v tom zmysle, že: *„Na didaktike si každý môže povedať svoj názor, aj tak bude mať pravdu.“* My sa domnievame, že v tomto prípade nastáva určité neporozumenie, ktoré vyplýva zo stretnutia dvoch rôznych svetov: sveta matematiky so svetom pedagogiky a psychológie. V matematike sme si zvykli na exaktné formulácie a formálne dokazovanie. Toto však momentálne v didaktike robiť nevieme. Nie je to preto, že by didaktici boli nešikovní, ale preto, že reálny svet a vzťahy v ňom sú natoľko komplikované, že pre väčšinu z nich nemáme algebraický popis. Tvrdenia napísané v prirodzenom jazyku zatiaľ nevieme formálne dokazovať (je otázne, či to vôbec niekedy budeme vedieť). Napriek tomu sme sa v didaktike posunuli ďalej, smerom k vedeckému uvažovaniu: máme k dispozícii výskumné metódy, ktorými vieme sformulovať teóriu, alebo potvrdiť či vyvrátiť názor – aj keď iba do určitej miery.

Budúci učitelia informatiky

Študenti, budúci učitelia informatiky, už úspešne absolvovali kurz programovania. Na predmet Didaktika programovania prichádzajú s pragmatickým očakávaním, že sa naučia učiť programovanie. Vzhľadom na to, že k svojmu štúdiu a so svojimi pedagógmi majú dobré skúsenosti, majú aj k didaktike programovania pozitívny prístup, bez nejakých pozorovateľných predsudkov. Týka sa to aj takých študentov, ktorí sa neplánujú v budúcnosti venovať učiteľskému povolaniu.

Študenti však majú značne skreslenú predstavu o žiakoch a výrazne preceňujú ich schopnosti. Pri vyučovaní chcú študenti postupovať rýchlym tempom a chcú riešiť ťažké príklady. Nezriedka ani neveria tomu, že niektoré úlohy, ktoré zvolili, môžu byť pre žiakov veľmi ťažké. Domnievame sa, že tento problém je spôsobený viacerými faktormi. Študenti si často idealizujú svoje spomienky – to, ako oni sami vnímali informatiku na základnej alebo strednej škole. Zabudli tiež na svoje problémy a trápenie sa na hodinách informatiky. Zabudli aj na negatívne vnímanie vyučovania ich vlastnými spolužiakmi. Navyše sa ukazuje, že štúdium na vysokej škole výrazne posunulo ich vnímanie toho, čo je ľahké. Naším cieľom je potom spomaliť tempo a usmerniť nároky takéhoto unáhleného budúceho učiteľa informatiky.

Pozorujeme, že vzorom pre viacerých študentov je bývalý učiteľ informatiky, od ktorého preberajú vyučovacie metódy: *„Nás to učili takto...„* Zatiaľ ale nerozumejú tomu, prečo ich učiteľ viedol vyučovanie takým spôsobom. Potom sa stáva, že musíme naprávať aj zlé návyky, ktoré chcú študenti prenášať do svojho vyučovania (napríklad, organizáciu vyučovacej hodiny, výber tém, názor na učebnice a pod.).

Niektorí zo študentov začínajú popri štúdiu na fakulte učiť informatiku na základnej alebo strednej škole (napríklad, učia prvý rok niekoľko hodín). Tým, že už pracujú so žiakmi a majú skúsenosti s vyučovaním, skôr porozumejú rôznym aspektom vyučovania. K vyučovaniu zatiaľ pristupujú veľmi intuitívne – a väčšinou skúšajú, čo ich žiaci ešte zvládnu absorbovať. Často však od lektorov veľmi ťažko akceptujú, že ich vzdelávacie ciele, alebo spôsob učenia nie sú najvhodnejšie. Zatiaľ sú totiž nadšení z toho, že ich žiaci počúvajú a rešpektujú, čo si myslíme, že je výborné. Pozorujeme však, že sa nedokážu sebakriticky pozrieť na svoje vyučovanie. Nemajú ešte vybudovaný nadhľad na to, čo je pre väčšinu žiakov akceptovateľné, čo bude pre nich prínosné aj z výhľadom do budúcnosti. Diskusie s týmito študentmi bývajú potom veľmi náročné.

Pre študentov je veľmi ťažké správne si zostaviť vlastnú vyučovaciu hodinu. Ak majú prakticky pred ostatnými spolužiakmi predviesť, ako by mala vyzerať ich vyučovacia hodina, dopúšťajú sa množstva chýb. Ich učenie má rýchle tempo, použijú prekomplikovanú motiváciu, nerozlišujú ľahké a ťažké úlohy. Zaujímavé však je, že pokým hodnotia vystúpenia svojich spolužiakov, dokážu tieto chyby veľmi dobre rozpoznať. Zdá sa teda, že teórii poznávacieho procesu rozumejú, pretože dokážu veľmi dobre kritizovať vystúpenia svojich spolužiakov. Chýba im však sebareflexia.

Učitelia z praxe, ktorí ešte nezažili didaktiku programovania

S učiteľmi z praxe sa stretávame v rámci ich celoživotného vzdelávania – na rôznych školeniach, konferenciách, seminároch a kurzoch. Pracovali sme aj s učiteľmi, ktorí majú vo svojej aprobácii vyučovanie informatiky, ale aj s takými, ktorí si aprobáciu pre vyučovanie informatiky dorábajú.

Z pohľadu lektora je práca s takýmito učiteľmi veľmi náročná:

- Mnohí učitelia sa na nás (ako lektorov) pozerajú s dešpektom: „čo nám idete vy, z vysokej školy hovoriť o učení, keď ani neviete, čo sa na základnej a strednej škole deje“. Pritom nevedia, že na základnej alebo strednej škole sme aj my učili.
- Učitelia nepoznajú cieľ didaktiky. Pravidelne sa stáva, že účastníci vzdelávania sa v rámci didaktiky programovania snažia riešiť problémy s nedostatkom učební, ekonomickú situáciu školy (zastarané počítače) alebo problémy s rodičmi.

- Pravdepodobne viacerí z nich mali zlú skúsenosť s didaktiku iného predmetu (napríklad, hrávali sa rôzne spoločenské psychologické hry), takže spočiatku pristupovali k didaktike programovania s predsudkami až s pohŕdaním.

Vo viacerých učiteľoch sa však miešajú negatívne aj pozitívne očakávania. Napríklad, prichádzajú s očakávaním, že spoznajú metodiku, ako učiť programovanie.

Navyše, mnoho učiteľov ešte stále neverí, že programovanie je súčasťou informatiky. Potom didaktiku programovania považujú za predmet, ktorý nemá v ich vzdelávaní miesto. Napríklad, na jednom stretnutí v rámci vzdelávania učiteľov z praxe sme zažili takúto situáciu: v rámci prvej aktivity mali učitelia vyriešili jednoduchý programátorský problém, aby sa naladili na programovanie a neskôr sme mohli úlohu spoločne analyzovať z pohľadu didakticky programovania. Dve tretiny učiteľov však prekvapilo to, že sa od nich na didaktike programovania naozaj vyžadovali programátorské schopnosti. Tretina z nich bola dokonca natoľko zaskočená, že túto úlohu úplne odmietli riešiť a s nami spolupracovať.

Učitelia niekedy podceňujú význam didaktiky programovania a nevenujú jej dostatočnú pozornosť. Buď kvôli predsudkom k didaktike, alebo ju jednoducho podcenia, pretože sa im zdá všetko veľmi jednoduché. Tak sa stane, že neporozumejú tomu, čo je cieľom didaktiky, nepochopia kritériá dobrého vyučovania. Potom vystúpenie takých učiteľov, počas ktorého mali predviesť vzorovú vyučovaciu hodinu, obsahovalo množstvo didaktických chýb. Absolútne neporozumeli tomu, že sa majú riadiť určitými pravidlami, ktoré sme si spoločne dopredu stanovili – učili tak, ako ich to napadlo. Preto sa zdá, že je dôležité, aby sme učiteľom z praxe povedali, že: *„V škole môžete učiť podľa seba – v tom vám nikto nezabráni. Teraz ale ideme trénovať naše didaktické pravidlá. Preto sa zamerajte na to, aby ste ich pri svojom vystúpení precízne dodržali.“*

Skúsení učitelia zvyknú vo svojej pedagogickej praxi „uletieť“. Napríklad, po absolvovaní didaktiky programovania učiteľka prezradila, že *„Už som nevedela, čo mám so žiakmi robiť. Učila som už všetko možné a čím ďalej tým viac, ale žiakov to stále menej a menej bavilo. Po Didaktike programovania som si uvedomila, čo je dôležité a začala som učiť radšej pomalšie. A skutočne to žiakov bavilo.“*

Ak sú učitelia opatrne upozornení, že ich spôsob vyučovania je nevhodný, napríklad preto, lebo majú rýchle príliš tempo, reagujú: *„Moji žiaci sú šikovní a takto to zvládnu.“* Jednej učiteľke sme sa neskôr v diskusii spýtali, či by týmto spôsobom vedela učiť aj žiakov z iných ročníkoch. Odpovedala, že: *„toto sú výnimočne šikovní žiaci. Tí, ktorí idú po nich, sú už slabší. S nimi by sa to vôbec nedalo robiť“ …*

Učitelia nie sú ochotní pripustiť, že ich žiaci niečomu nerozumejú, alebo majú s niečím problémy. Iba výnimočne sa nájdu učitelia, ktorí sa na svoje vyučovanie pozerajú kriticky a prezradia, že žiaci aj niečo nezvládali. Väčšina učiteľov však hovorí, ako ich *„žiaci fantasticky pracovali“*, ako sa im darilo. Domnievame sa, že je to dané rôznymi faktormi:

- Učitelia sa chcú pochváliť pred ostatnými kolegami, akých majú fantastických žiakov, alebo ako fantasticky učia.
- Učitelia nie sú naučení kriticky sa verejne vyjadrovať.
- Učitelia si netrúfnu zaujať k niečomu negatívny postoj (pravdepodobne sa necítia byť odborníkmi, alebo majú s danou témou málo skúseností).
- Učitelia sa obávajú toho, že budú vysmiati, ak pripustia, že práve ich žiaci mali s niečím problém.

Počas práce s učiteľmi sa ukázalo, že didaktika programovania pokrýva už natoľko širokú vekovú škálu žiakov, že je potrebné rozčleniť ju zvlášť pre učiteľov, ktorí učia na základnej škole a zvlášť pre učiteľov, ktorí učia na strednej škole.

V učiteľskej verejnosti prevažuje väčšinou neznalosť, prípadne až negatívna predstava o didaktike programovania. Ľudia, ktorí sa k didaktike stavajú nezaujato až pozitívne, sú väčšinou naši študenti, budúci učitelia informatiky. Mrzuté je, že učitelia z praxe často nezažili dobrý didaktický predmet. Majú teda k didaktike programovania negatívny postoj. Potom je našou úlohou, aby sme im ich zlú mienku o didaktike napravili.

IV. Obsah seminárov

Didaktiku programovania chceme realizovať, pokým sa to dá, konštuktivistickým spôsobom. Znamená to, že sa snažíme minimalizovať množstvo teoretických informácií, a naopak, čo možno najviac zapájať do činnosti našich študentov.

Realizujeme aktivity z rôznych oblastí, ale prevažne takomto poradí:

- Aktivity, ktorých cieľom je oboznámiť sa s postavením programovania v školách. Študenti sa popritom učia orientovať sa v pedagogických dokumentoch. Zároveň spoznávajú slovník, ktorý sa v týchto dokumentoch používa.
- Aktivity, ktoré sú zamerané na didaktický pohľad na programovacie jazyky. V aktivite nepredkladáme študentom správne názory. Otázkami ich však vedieme k tomu, aby sami rozmýšľali o programovacích jazykoch a ich výhodách/nevýhodách. Výber jazykov, kritérií a ich výsledné hodnotenie potom významne závisí od konkrétnej skupiny študentov.
- Aktivity, ktorá sú zamerané na analýzu učebníc. Študenti sa zároveň učia rozpoznávať jednotlivé etapy teórie poznávacieho procesu.
- Aktivity, ktoré sú zamerané na rôzne prístupy k vyučovaniu programovania. Tu zároveň vidno komplexnosť toho, ako je vyučovanie náročné na zvažovanie rôznych alternatív a prístupov.
- Aktivity, ktoré sú zamerané na nácvik vyučovania. Ich cieľom je, aby sa študenti učili rozmýšľať o (svojom) vyučovaní.
- Aktivity, ktoré sú zamerané hodnotenie žiakov z programovania. Ich cieľom je, aby si študenti spoznávali objektívne a neprestrelené kritéria hodnotenia.

Ukázalo, že je dôležité zoradiť aktivity smerom od strednej školy k nižším stupňom. Spočiatku, keď sme začínali viesť semináre z didaktiky, pozorovali sme, že naši študenti vôbec nerozumeli ani neverili problémom, ktoré súvisia s vyučovaním mladších žiakov. Až neskôr sme pochopili, že pravdepodobnou príčinou je obrovská časová priepasť medzi vysokoškolákmi a žiakmi zo základnej školy. Pre vysokoškolských študentov býva obrovským mentálnym problémom to, aby sa znížili k uvažovaniu mladších žiakov. Preto zoznamujeme študentov s vyučovaním často v opačnom poradí: od maturity, cez strednú školu až k nižším stupňom základnej školy.

Maturita

Cieľ: Spoznať náročnosť úloh z programovania, ktorá sa vyžaduje na maturite z informatiky.

Maturitu považujeme za cieľovú métu stredoškolského štúdia časti žiakov. My však didaktiku programovania maturitou začíname. Viedli nás k tomu nasledujúce dôvody:

- Chceme, aby sa študenti vžili do úlohy svojich budúcich žiakov a tiež aj to, aby sa oboznámili náročnosťou maturitnej skúšky z informatiky.
- Maturita z informatiky by mala vymedzovať obsah, ktorý považujeme za dôležitý z pohľadu informatiky ako vedného odboru.
- Je potrebné poznať, kam vyučovanie informatiky a programovania smeruje od základnej po strednú školu.
- Na maturitných úlohách sa dá vidieť, ktoré kompetencie (poznatky, schopnosti) z informatiky sú dôležité.

Domnievame, že s obsahom maturít by mal byť oboznámený nielen učiteľ informatiky strednej školy, ale aj učiteľ, ktorý učí na základnej škole. Tak aspoň pozná cieľ (aj keď dlhodobý a vzdialený), pre ktorý pripravuje a vzdeláva svojich žiakov.

Aktivita 1:

Vyriešte úlohy z maturitnej skúšky z informatiky.

Priebeh:

- Lektor vytlačí maturitné úlohy pre každého študenta.
- Študenti ich budú samostatne riešiť – 60 až 120 minút.
- Je možné, že študenti sa budú snažiť úlohy rôzne komentovať. Oplatí sa ich v tomto podporiť s tým, že: *„Je výborné, ak si všímate aj tieto veci. Skúste sa ale teraz nerozprávať nahlas. Robte si poznámky, po vyriešení úloh sa budeme o vašich postrehoch určite baviť.“*

Poznámky:

- Keďže momentálne neexistuje externá maturitná skúška z informatiky, používame staré prípravné testy, ktoré mali slúžiť ako vzor pre budúcu externú maturitnú skúšku z informatiky:

  `http://www.nucem.sk/documents//25/monitor_a_gs_2004/12_Test_I-2.pdf`

  Nevadí, že úlohy sú také staré. Otvára sa tým úžasný priestor na diskusiu so študentmi. Navyše, náročnosť úloho z programovania približne zodpovedajú náročnosti, ktorá sa očakáva aj na súčasných maturitných skúškach.
- Predpokladáme, že s maturitou v Českej republike je situácia podobná – momentálne sa mení rámcový vzdelávací program a ešte neexistujú vzorové maturitné zadania úloh ani testy. V tejto situácii odporúčame, aby lektori zohnali úlohy od učiteľov, ktorí pripravujú svojich žiakov na maturitné skúšky z informatiky a robia aj programovanie.

Riešenie maturitných úloh sa nám zdalo ako vhodný prechod od programovania, ktoré študenti zažívali v rámci vysokoškolského štúdia, k stredoškolskému programovaniu. Na vysokej škole sa učia veci, ktoré sú výrazne nad rámec strednej školy, a potom študenti strácajú cit aj pochopenie pre stredoškolskú úroveň informatiky. Riešením maturitných úloh sa postupne nastavia na úroveň, ktorá je vyžadovaná na strednej škole, na konci štvrtého maturitného ročníka. Šikovným študentom sa potom zdá, že maturitný test je veľmi-veľmi jednoduchý, slabým študentom sa zdá

príliš náročný. Každopádne sa tým pre študentov definuje určitá hranica, od ktorej sa môže lektor odraziť.

Aktivita 2:

Ako sa vám páčili maturitné úlohy?

Diskutujte o:

- dojmoch z riešenia,
- zadaniach a úlohách,
- do ktorej úrovne Revidovanej Bloomovej taxonómie by ste jednotlivé úlohy zaradili,
- ich jednoznačnosti, prípadne zmysluplnosti (s odstupom niekoľkých rokoch).

Priebeh:

- Aktivita nadväzuje na predchádzajúcu aktivitu. V diskusii sa snažíme aj o to, aby sa študenti naučili rozlišovať a pomenovať jednotlivé informatické koncepty, ktoré sa v každej úlohe testujú, ale aj na spôsob ich testovania.
- Tu zvykneme zapnúť projektor, premietnuť zadania a diskutovať so študentmi o postupne každej úlohe.

Úlohy v maturitnom teste testujú rôzne kompetencie (schopnosť porozumieť cudziemu algoritmu, schopnosť trasovať program, ...), robia to rôznym spôsobom (treba zistiť, čo program robí, treba doplniť program tak, aby čosi vykonal,...), a tiež sú zamerané na rôzne oblasti alebo témy (cyklus, polia, náhodné čísla a podobne). Počas diskusie v Aktivite 2 sa študentov pýtame:

- *„Pomenujte tému alebo oblasť z programovania, ktorej sa jednotlivé úlohy týkajú.“*
- *„Povedzte, aké kompetencie a akým spôsobom jednotlivé úlohy testujú.“*

Predchádzajúca aktivita nezriedka pozvoľna prejde do diskusie o štátnom vzdelávacom programe, postavení a cieľoch informatiky a programovania vo vzdelávaní.

Tip: V prípade, že kurzu didaktiky programovania robíme pre učiteľov z praxe, ktorí učia iba na základnej škole, odporúčame v Aktivite 1 namiesto riešenia úloh z maturitného ročníka riešiť úlohy, ktoré zvládajú žiaci na konci 2. stupňa základnej školy. V rámci Aktivity 2 potom diskutujeme o týchto úlohách, zadaniach atď. K maturite a štátnym dokumentov sa ešte dostaneme v rámci nasledujúcich aktivít.

Ako učiť riešiť algoritmické problémy

Cieľ: Zoznámiť študentov s tým, že k riešeniu programátorskej úlohy vedie veľa drobných krokov. Zároveň to bude pre študentov jemný úvod do didaktiky programovania.

Mnohí študenti sú v programovaní vynikajúci a majú s programovaním dlhoročné skúsenosti. Preto mnohé úlohy riešia akoby v automatickom režime. Napríklad tak, že rovno napíšu výsledok, a kroky, ktoré k riešeniu vedú, si už ani neuvedomujú. Je to podobné, ako keby sme sa opýtali: *„Koľko je 100 – 50?“* Pravdepodobne by sme okamžite odpovedali, že výsledok je 50. Žiaci sa však odčítavanie učia na matematike niekoľko hodín a ďalej ho trénujú aj niekoľko rokov.

Aktivita 3:

Predstavme si na chvíľu, že sme na gymnáziu. Žiaci sa naučili používať premenné a cyklus. Tie majú natrénované a veľakrát ich už používali pri riešení úloh na predchádzajúcich hodinách. Teraz chceme učiť riešiť problémy, v ktorých treba výsledok akumulovať. Žiakom sme zadali úlohu:

„Prefíkaný obchodník predával svoj tovar tak, že nenápadne zvyšoval jeho cenu. Pri prvom nákupe sme zaplatili 1 korunu. Pri každom ďalšom nákupe si však od nás vypýtal o 1 korunu viac, ako naposledy. Koľko by sme mu zaplatili, ak by sme uňho nakupovali 10krát?“

Ako by ste postupovali na hodine, ak by žiaci tento problém nedokázali sami vyriešiť?

Žiakom nechceme predložiť hotové riešenie, ale chceme ich na riešenie naviesť.

Úloha sa dá vyriešiť a zapísať napríklad v jazyku Python takto:

```
s = 0
for i in range(1, 11):
    s = s + 1 + i
```

Poznamenajme, že úloha sa dá vyriešiť aj takto, ale takéto riešenie od žiakov neočakávame:

```
s = sum(range(11))
```

Priebeh:

- Pri tejto aktivite si lektor vymení úlohu so študentmi. Z lektora sa stane žiak, ktorý ničomu nerozumie. Študenti sa stanú učiteľmi a snažia sa lektora naviesť na riešenie.
- Lektor bude písať na tabuľu tak, akoby si žiak rozmýšľal na papieri. Zároveň, keby Študenti vyžadovali od lektora veľké skoky v uvažovaní, bude na študentov reagovať tak, že tomu „N*erozumiem“*.
- Pre študentov bude dôležité, aby videli, že riešenie takej úlohy sa skladá z množstva drobných krokov a nie triviálnych úvah.

S reálnymi žiakmi by sme najskôr o úlohe diskutovali. Overili by sme si napríklad, či porozumeli generujúcej formulácii tak, že sa ich opýtame: *„Koľko zaplatíme za prvý nákup?“*, *„Koľko za druhý?“*, *„Tretí?“* , *„A za posledný?“*.

Pýtali by sme sa ďalej: *„Akú sumu zaplatíme za všetky nákupy dohromady?“*. V tomto prípade má úloha naschvál triviálne riešenie, takže žiaci nám rýchlo odpovedali, že vlastne potrebujeme sčítať čísla:

```
1 + 1 + 2 + 3 + ... + 10
```

Pokračovali by sme teda ďalej: *„Predstavte si ale, ako by program vyzeral, keby sme sa rozhodli sčítať 1000 čísel. S písaním by sme sa poriadne natrápili. Dá sa to naprogramovať nejako elegantnejšie?“* Žiaci by sa teraz ocitli na hranici svojich možností.

Pomohli by sme im analógiou: *„Ako by sme tieto čísla sčítali bez počítača?“*.

Zrealizovali by sme niekoľko krokov: *„Najskôr k číslu 0 pričítame 1, potom pripočítame 2, potom 3, a tak ďalej.“* Popritom by sme na tabuli znázorňovali jednotlivé kroky, čísla a súčty. Bolo by dôležité, aby žiaci videli, že výsledok vzniká postupne, nie v jednom okamihu. Okrem toho by sa začala objavovať schéma: *„Vidíme, že súčet vzniká postupne. K predchádzajúcemu výsledku pripočítame číslo. Tak vznikne nový súčet.“*

A ďalej: *„Ten si musíme zapamätať, inak by sme nevedeli v sčítavaní pokračovať. My sme si výsledok písali na tabuľu. Náš program si ho zapamätá pomocou premennej. Nazvime ju napríklad* `s`*.“* Bolo by dôležité, aby žiaci zažili okamih, keď vzniká potreba použitia premennej a rozumeli tomu, na čo premenné vo výslednom programe poslúži.

Naše zápisy na tabuli boli zatiaľ neformálne. Teraz by sme ich priblížili k výslednému programu: *„Na začiatku sme ešte nič nesčítali. Preto po spustení programu nastavíme premennú s na nulu.“* Vysvetľovali by sme a zároveň kreslili na tabuľu:

```
s = 0
```

Známymi príkazmi by sme zapisovali aj ďalšie kroky a slovne by sme ich komentovali: *„Postupne pripočítame jednotlivé čísla:“*

```
s = s + 1          „Po vykonaní tohto príkazu bude v premennej s hodnota 1.”
s = s + 2          „V premennej s bude hodnota 3.”
s = s + 3          ... s = 6
s = s + 4          ... s = 10
...
```

Vsunuli by sme ešte jeden všeobecnejší krok: *„Ako by sme k súčtu pripočítali nejaké číslo* `i`*?“*

```
s = s + i
```

Nakoniec, by bolo dobré, aby žiaci videli aj hraničnú situáciu: *„Ako k* `s` *pripočítame posledné číslo?“*

```
s = s + 10
```

Takéto krokovanie a naše zápisy na tabuli by boli veľmi dôležité kvôli tomu, aby sme žiakom poskytli príležitosť na objavenie opakujúceho sa vzoru: *„Prvý príkaz* `s = 0` *je špeciálny. Ale ostatné vyzerajú podobne:* `s = s + cosi`*. A to* `cosi` *sa mení od 1 po 11.“*

Keďže naši žiaci už poznali príkaz cyklu a jeho používanie mali perfektne natrénované. Mohli by sme sa ich opýtať: *„Ako zabezpečíme, aby sa v programe niečo menilo od 1 po 10?“*

Tak by sme sa dopracovali k zápisu:

```
s = 0
for i in range(1, 11):
    s = s + i
```

Poznamenajme ešte, že nie vždy by sme museli so žiakmi realizovať takto detailný postup. Niektorým by stačilo riešenie iba naznačiť, a potom by už celý algoritmus sami domysleli aj naprogramovali. Naopak, s inými by bolo potrebné prejsť všetky kroky pre všetky čísla.

Aktivita 4:

Vyberte si jednu z nasledujúcich programátorských úloh a vyriešte ju. Vysvetlite ideu riešenia svojom spolužiakom.

Každá skupina si zvolí zástupcu, ktorý vystúpi pred ostatnými kolegami:

- Vysvetlite, ako ste prišli na jednotlivé príkazy, konštrukcie, premenné a všeobecné výrazy (resp. ako by mali žiaci postupovať pri ich objavovaní).
- Ideálne vystúpenie by malo trvať najviac 5 minút

Jednotlivé vystúpenia pozorujte a robte si k nim poznámky.

Úlohy:

A) Vytvorte program, ktorý nakreslí hviezdnu oblohu tak, že na modré pozadie sa nakreslí 1000 náhodne rozmiestnených bielych bodiek.

B) Poznáte príbeh o prešibanom mudrcovi a šachovi, ktorý sľúbil za odmenu toľko zrniek, koľko sa zmestí na políčka šachovnice? Presnejšie tak, že na prvé políčko dáme jedno zrnko, a na každé ďalšie políčko dáme dvojnásobný počet zrniek? Pomôžte šachovi v rozhodovaní, či je taká odmena adekvátna a vytvorte preňho jednoduchý program. Do programu zadáme rozmery šachovnice a ten nám spočíta a povie, koľko zrniek by malo byť dohromady na šachovnici. Napríklad, pre vstup `2` (šachovnica `2x2`) zobrazí výsledok `15 (= 1 + 2 + 4 + 8)`.

C) V lesnej škôlke rastú stromčeky. Zatiaľ sú malé a majú náhodnú výšku aj šírku – od 30 do 100 cm. Nakreslite 10 z nich na obrazovku vedľa seba tak, aby sa neprekrývali. Koruna stromu je zelený kruh a kmeň hnedý obdĺžnik.

D) Vytvorte program, ktorý bude generovať umelecké farebné mozaiky. Kamienok mozaiky je štvorec so stranou 10 a mozaika má rozmery 20 štvorcov na šírku a 10 na výšku. Každý kamienok je náhodne zafarbený.

E) V krajine Hromovo je stále zamračené a prší. Aby sme potešili jej obyvateľov, rozhodli sme sa vytvoriť program, ktorý nakreslí slniečko. Slniečko je žltý kruh a z neho vychádza napríklad 25 slnečných lúčov.

F) Poverčivému kráľovi sa prisnilo, že šťastie mu prinesú iba magické čísla. Preto si zavolal svojho dvorného počtára, aby mu také čísla našiel. Vraj to musia byť trojciferné čísla, ktorých súčet cifier je deliteľný 7. Aké čísla to boli? A koľko ich bolo? Pomôžte dvornému počtárovi a napíšte program, ktorý to zistí.

G) Viete ako vyzerá pečiatka, ktorá označuje utajené dokumenty? Aj my chceme vytvoriť tajnú aplikáciu a označiť ju podobnými pečiatkami tak, že štyri pečiatky budú zobrazené v rohoch a jedna pečiatka v strede grafickej plochy. Pečiatka by mohla mať tvar červeného obdĺžnika, v ktorom by bolo napísané „Prísne tajné!“.

H) Prvočíselné dvojčatá sú také dve prvočísla, medzi ktorými je rozdiel 2. Napríklad (3, 5), (5, 7), (11, 13), ... Sme zvedaví, koľko existuje prvočíselných dvojčiat medzi číslami od 1 po 1000. Vytvorte program, ktorý ich vypíšte a nakoniec zobrazí aj ich počet.

I) Pozerali ste sa niekedy do farebného kaleidoskopu? Vytvorte program, ktorý nám umožní kresliť pomocou myši podobné symetrické a farebné obrázky. Pri pohybe myšiu sa nakreslí pod kurzorom náhodne zafarbená bodky. A ďalšie 3 bodky sa nakreslia symetricky podľa vodorovnej a zvislej osi, ktoré prechádzajú stredom grafickej plochy.

J) Autobus jazdí medzi dedinami Hornou a Dolnou. Na tejto trase je 10 zastávok. Na každej z nich nastupovalo a vystupovalo niekoľko ľudí. Počet ľudí, ktorý pribudol alebo ubudol v autobuse na jednotlivých zastávkach je uložený v 10 prvkovom poli/zozname. Zistite, medzi ktorými zastávkami bolo v autobuse najviac ľudí. Napríklad, pre prvky:

```
   5     -2     8     13     -15     7     9     -12     3     -16
```

program vypíše, že `Najviac ľudí cestovalo medzi 7. a 8. zastávkou.`

Priebeh:

- Lektor úlohy vytlačí na papieriky. Študenti si ich vylosujú, ale ak sa im úloha nepáči, môžu si ju vymeniť.
- Študentom necháme čas na vyriešenie úlohy. Môžu si zvoli jazyk aj prostredie, ktoré im vyhovuje.
- Úlohy môžu riešiť na papier alebo pomocou počítača.
- Pri vysvetľovaní **nebudú nepoužívať počítač ani projektor.**
- Je dobré, aby sa študenti učili **kresliť a znázorňovať na tabuľu**.

Ak je študentov viac, rozdelíme ich do skupín. Ak je študentov menej, niektoré úlohy ostatnú nevyriešené. Máme skúsenosti, že táto aj nasledujúca aktivita je zaujímavá aj pre študentov aj pre učiteľov z praxe. Úloh je však veľmi veľa. Preto sme ich zvykli realizovať aj na dvoch seminároch – jednej polovici úloh sme sa venovali na jednom a druhej polovici úloh na druhom seminári.

Aktivita 5:

Je dobré, ak aktivity vyvolajú diskusiu. Môžeme ju podporiť otázkami:

- Bolo riešenie správne?
- Čo by ste navrhli zlepšiť, vymeniť?
- Čo sa vám páčilo a prečo?

Priebeh:

- Ak treba, lektor sa pýta: *„Odkiaľ sa zobral tento vzorec? Na čo slúži premenná? Prečo je tu cyklus? Ako to vzniklo?“* Počas tejto hry si študenti uvedomia, že naviesť žiaka na riešenie problému vôbec nie je triviálne.
- Po čase sami študenti budú svojho spolužiaka upozorňovať na príliš veľké skoky v úvahách, alebo sa budú pýtať, ako by to lepšie vysvetlil svojim žiakom.

Programovanie v osnovách

Cieľ: Zoznámiť sa s vývojom vyučovania informatiky a porozumieť postaveniu programovania v rámci základoškolského a stredoškolského vyučovania informatiky.

Aktivita 6:

Ako si spomínate na informatiku na základnej a strednej škole?

Priebeh:

- Lektor postupne vyzýva študentov, aby povedali svoje skúsenosti zo základnej a strednej školy. Podporuje diskusiu medzi spolužiakmi – každý má inú skúsenosť, iné zážitky.
- Diskusiu môže podporiť otázkami:
  „Učili ste sa aj programovanie?“
  „Kedy ste začali učiť programovať?“
- Prípadne, ak niektorí zo študentov už popri svojom štúdiu učia, je dobré sa ich opýtať:
  „Ako vyzerá informatika na vašej škole?“
- Je dobré, ak sa lektor na záver podelí aj o svoje skúsenosti – napríklad, ako vyzeralo vyučovanie informatiky 80. alebo 90. rokoch.

Aktivita 7:

Preštudujte *Rámcový vzdelávací program* pre jednotlivé stupne základnej a strednej školy pre informatiku aj dokument o maturite z informatiky. Zamerajte sa iba na programovanie a algoritmické myslenie. Diskutujte o obsahovom a výkonovom štandarde – čo sa očakáva od programovania na jednotlivých stupňoch a na maturitnej skúške?

Priebeh:

- Lektor postupne premietne dokumenty a diskutuje so študentmi o pojmoch a výkonoch, ktoré sa očakávajú od žiakov.
- Tiež je zaujímavé pýtať sa a trénovať študentov:

  „Do ktorej úrovne Revidovanej Bloomovej taxonómie by ste výkon zaradili?“
- V štátnych dokumentoch sa používa abstraktné formulácie. Lektor by sa mal preto občas opýtať a otázkami vyvolať diskusiu:

  „Čo si máme pod tým predstaviť?“

Študenti nemusia rozumieť formuláciám v rámcových plánoch – je to pochopiteľné, lebo si pod nimi nedokážu nič zmysluplné predstaviť. Ak na predchádzajúcu otázku nezvládnuť odpovedať, je dobré, ak im lektor pomôže. Pozor ale, aby sa diskusia nezmenila na prednášku.

Aktivita 8:

Pozrite sa na hodinové dotácie určené pre informatiku. Stačia na vyučovanie informatiky?

Tipy:

- Je zaujímavé, ak hodinové dotácie porovnávame s inými predmetmi. Väčšine študentov sa zadajú veľmi nízke.
- V prípade, že kurzu didaktiky programovania robíme pre učiteľov z praxe, odporúčame otvoriť diskusiu, ktorej cieľom je výmena skúseností z rôznych škôl:
  „Ako ste si poradili s hodinovými dotáciami na vašej škole?“

Aktivita 9:

Pre každý stupeň základnej školy aj pre strednú školu vymyslite a navrhnite 3 úlohy alebo príkladu z programovania tak, aby spĺňali obsahový a výkonový štandard príslušného stupňa.

Priebeh:

- Študenti opäť pracujú v skupinách, tak ako boli rozdelení v predchádzajúcej aktivite.
- Na konci prezentujú a obhajujú svoje návrhy. Lektor sa k nim vyjadruje, prípadne otvára diskusiu.

Aktivita 10:

Uvažujte nad otázkou: „Prečo je programovanie dôležitou časťou informatiky na ZŠ, SŠ?“. Dôvody povedzte.

Aktivita 11:

Čo si myslíte o algoritmoch a programovaní? Sú to rovnaké veci alebo sa líšia? V čom?

Musíme dopredu upozorniť, že pojem programovanie používame v posunutom zmysle slova. Totiž, klasické definície pojmov „algoritmus“ a „programovanie“ hovoria zhruba nasledovné:

- Algoritmus = postup, návod na riešenie problému.
- Algoritmizácia = vytváranie (vymýšľanie a navrhovanie) algoritmov.
- Programovanie = prepis algoritmu do programovacieho jazyka.

Takéto striktné oddeľovania návrhu algoritmu od jeho prepisu do programovacieho jazyka malo v minulosti zmysel, pretože výpočtový čas bol drahý a neexistovali ani vhodné editory. V súčasnosti však takéto striktné oddeľovanie algoritmov od programovania stráca zmysel. Súčasné vývojové prostredia nám nielenže umožňujú algoritmy zapisovať priamo v programovacom jazyku, ale pomáhajú nám ich ladiť, vylepšovať (napr. profiler) alebo upravovať (napr. refaktorizácia).

Pre konštruktivistický prístup pri vyučovaní považujeme za nesmierne dôležité to, že naše nápady môžeme okamžite vyskúšať. Napríklad, v interpretovanom jazyku Python píšeme príkazy do príkazového riadku. Výsledok vykonávania príkazov vidíme ihneď po stlačení klávesu Enter na obrazovke. Ešte ďalej ide v tomto smere prostredie EasyLogo. V ňom skladáme program z kartičiek, pričom zmena programu alebo parametra príkazu sa okamžite prejaví na výslednom obrázku. Vďaka takémuto experimentovaniu získavajú žiaci spätnú väzbu a cenné skúsenosti s tým, ako jednotlivé príkazy fungujú.

Často sa v súvislosti s pojmom algoritmu vynárajú aj jeho vlastnosti: *jednoznačnosť*, *konečnosť*, *rezultatívnosť*. Veľa učiteľov práve týmito vlastnosťami a „definíciami“ začína svoje vyučovanie programovania. To je ale zlozvyk.

My napríklad za algoritmus považujeme aj taký návod, ktorému nemusíme rozumieť. Dôležité je, že má vykonávateľa, ktorý dokáže podľa neho postupovať. Sú mnohé algoritmy, ktoré pracujú s náhodnými číslami, takže o jednoznačnosti je v ich prípade naozaj ťažké hovoriť. Program, ktorý riadi svetelnú križovatku alebo operačný systém, ktorý by mal fungovať do nekonečna, sú tiež algoritmy. Pritom nespĺňajú uvedené vlastnosti.

Na druhej strane, z pohľadu didaktiky, hovoriť deťom o vlastnostiach algoritmov skôr, ako napíšu akýkoľvek program, nemá zmysel. Žiaci týmto vlastnostiam v skutočnosti nebudú rozumieť. Pre

žiakov to budú informácie, ktoré sa musí nabifľovať, aby potešil učiteľa, ktorý ho vyskúša. Žiak mu porozpráva o jednoznačnosti, a pritom o pár hodín sa bude učiť používať generátor náhodných čísel...

Hranica medzi tvorbou algoritmov a programovaním sa stráca. V súčasnosti, najmä pri vyučovaní programovania, považujeme za dôležité, aby žiaci čo možno najskôr zapisovali a overovali si algoritmy vo vhodnom prostredí na počítači. Už nepovažujeme za potrebné učiť začiatočníkov tak, ako kedysi, rozmýšľať a zapisovať programy v špeciálnom „algoritmickom“ jazyku:

- Takýto jazyk mával často textovú podobu. Pritom vychádzal z prirodzeného jazyka s rôznymi frustrujúcimi dodatočnými obmedzeniami. Napríklad, pri hľadaní najnižšej paličky metódou „pozriem sa a vyberiem najnižšiu“ učiteľ zakáže operáciu „pozretia“ z pre žiakov nejasných dôvodov.
- Zvykli sa používať aj rôzne grafické zápisy, napríklad vývojové diagramy alebo štruktúrogramy. Pri ich zápise už bolo potrebné dodržiavať určité formálne pravidlá. Preto sa do určitej miery dajú tieto zápisy chápať ako programy, ktoré však nemáme šancu spustiť, vyskúšať.

Tieto prístupy dnes považujeme za prekonané. Algoritmické jazyky tvoria zbytočnú medzivrstvu medzi nápadom, programovaním a výsledkom. V konečnom dôsledku tak odďaľujú etapu zbierania skúseností.

Prostredia pre vyučovanie programovania

Cieľ: Spoznať rôzne prostredia, ktoré sú určené na vyučovanie programovania a porozumieť ich určeniu v kontraste s profesionálnymi programovacími jazykmi a vývojovými prostrediami.

Aktivita 12:

V čom sa líšia nasledujúce úlohy, hoci na prvý pohľad sú veľmi podobné:

- Máme štvorec so stranou dĺžky `a`, chceme nakresliť kružnicu doň **vpísanú**.
- Máme štvorec so stranou dĺžky `a`, chceme kružnicu okolo neho **opísanú**.

Priebeh:

- Študenti sa zamyslia nad riešením a pokúsia sa vysvetliť rozdiely.
- Lektor upozorní na to, že pri takýchto úlohách dobre vidno význam grafického výstupu.

Väčšina grafických knižníc kreslí kruhy tak, že parametrami určíme obdĺžnikovú oblasť, do ktorej sa vpíše elipsa. Stačí teda vypočítať súradnice, čo je v prvej úlohe ľahké, ak `x, y` zodpovedá ľavému hornému rohu štvorca:

```
canvas.create_oval(x, y, x + a, y + a)
```

Druhá úloha je matematicky náročná, lebo treba počítať polomer kružnice pomocou odmocnín. Navyše, ak `x, y` predstavuje ľavý horný roh štvorca, a nie jeho stred, odvodenie vzorcov pre výpočet parametrov príkazu `create_oval` sa veľmi skomplikuje.

Stalo sa nám, že žiak polomer nastavoval metódou pokus – omyl. Namiesto presného vzorca

```
r = a / (2 ** 0.5)
```

použil vzorec

```
r = a * 3 / 4
```

.

Je úžasné, že použil taký postup, ale bol sklamaný, pretože vzorec dobre fungoval pre malé a (napríklad, pre `a = 10` vychádza v oboch prípadoch približne rovnaký výsledok `7`), ale pre väčšie čísla, napríklad `a = 100`, už kružnica neprechádzala cez vrcholy štvorca. Grafický výstup tu otvoril priestor experimentovaniu, ale zároveň poslúžil ako jednoduchá skúška správnosti. Keby sme nechceli kružnicu nekreslil, ale iba vypísať výsledok, správnosť vypísanej hodnoty by sa ťažšie skontrolovala.

Pre vyučovanie je dôležité poznať výhody a nevýhody programovacích jazykov aj vývojových prostredí. Ak sme si vedomí problémov, ktoré nás očakávajú, alebo ich vieme dopredu odhadnúť, môžeme lepšie uvažovať o tom, ako sa im pri vyučovaní programovania v škole vyhnúť. Výhody zase môžeme využiť vo svoj prospech.

Aktivita 13:

Preskúmajte rôzne programovacie jazyky a prostredia, napríklad:

- FreePascal
- Delphi alebo Lazarus
- Python
- C, C++ a Visual Studio
- Java v Eclipse, Android Studio alebo inom prostredí
- Visual Basic napríklad v tabuľkovom kalkulátore

Priebeh:

- Lektor v krátkosti ukáže, ako jednotlivé prostredia vyzerajú – ukazuje také, s ktorými ma aspoň drobné skúsenosti. Môže mať pripravené ukážky toho, ako v každom jazyku vyzerá program, ktorý po spustení pomocou cyklu zobrazí čísla od `1` po `10`.
- Ak zvláda, môže v krátkosti porozprávať o programovacích jazykoch – história, dôvod vzniku, ako sa ktorý jazyk inšpiroval od iného jazyka.

Na internete existuje veľa diagramov, ktoré znázorňujú historický vývoj jazykov:

```
https://www.cs.toronto.edu/~gpenn/csc324/PLhistory.pdf
http://rigaux.org/language-study/diagram.html
```

Študentom zvykneme taký graf premietnuť a krátko ho spoločne preskúmať. Občas bývajú sami prekvapení, aké historické korene má ich obľúbený programovací jazyk.

Aktivita 14:

Ktoré vlastnosti programovacích jazykov alebo prostredí sú pre vás dôležité? Ktoré z nich považujete za dôležité pre profesionálnu tvorbu programov, a ktoré pre vyučovanie programovania?

Priebeh:

- Je dobré, ak si študenti sami zvolia kritériá (syntaktické pravidlá, zmysluplnosť chybových hlásení, podpora grafiky, rýchlosť, …) – lektor ich zapisuje pod seba.
- Táto aktivita nadväzuje na nasledujúcu aktivitu.

Aktivita 15:

Porovnajte programovacie jazyky z hľadiska ich vhodnosti pre vyučovanie programovania: Zostavte tabuľku, v ktorej • jeden rozmer tvoria vami stanovené kritériá pre vyučovanie z predchádzajúcej aktivity, • druhý rozmer tvoria vám známe programovacie jazyky alebo prostredia. Tabuľku vyplňte. Ktorý jazyk alebo prostredie spĺňa najviac kritérií a ktorý najmenej?

Priebeh:

- Študenti vymenujú jazyky, ktoré poznajú – lektor ich zapisuje vedľa seba.
- Vznikne dvojrozmerná tabuľka. Políčko v tabuľke bude určené na hodnotenie toho, ako daný programovací jazyk napĺňa stanovené kritérium. Políčka spoločne so študentmi vyplnia – do políčka môžu písať značku, známku alebo číselné body.
- Následne môžu diskutovať o tom, ktorý z jazykov spĺňa kritéria pre profesionálov, a ktorý pre vyučovanie

Detské programovacie jazyky a prostredia musia rešpektovať nielen určité ergonomické kritériá, ale bezpodmienečne musia rešpektovať aj schopnosti detí v danom veku. Vieme napríklad, že deti do 12 rokov sú schopné rozmýšľať a vykonávať operácie s konkrétnymi predmetmi (tzv. obdobie konkrétnych operácií). Abstraktné myslenie (tzv. obdobie formálnych operácií) nastupuje a rozvíja sa až v neskoršom veku. Detské prostredia preto vyzerajú veľmi odlišne od prostredí, ktoré sú určené pre dospelých a profesionálnych programátorov.

Aktivita 16:

Pozrite si (prípadne vyskúšajte) niektoré prostredia, napríklad:

- Cirkus šaša Tomáša
- Karel, Baltík, Panák
- Živý obraz
- Easy Logo
- Imagine
- Scratch

Pritom sa zamyslite a povedzte:

- Pre aký vek sú jednotlivé prostredia vhodné?
- Prečo prostredia vyzerajú tak, ako vyzerajú?
- Akým jazykom komunikujú s používateľom?
- Čo je cieľom prostredia – aké informatické kompetencie rozvíjajú (objavovanie algoritmu, porozumenie pravidlám, ovládanie objektov v priamom režime, zostavovanie poradia príkazov, trasovanie programu, riešenie elementárnych grafových alebo kombinatorických úloh, ...)?

Priebeh:

- Lektor v krátkosti ukáže, ako jednotlivé prostredia vyzerajú – ukazuje také, s ktorými má skúsenosti. Môže zvoliť aj iné.
- Ďalej diskutuje so študentmi o význame prostredia.

Aktivita 17:

Porovnajte jednotlivé prostredia:

- Aké metafory používajú jednotlivé detské prostredia?
- Aké koncepty tieto metafory zastupujú, predstavujú?

Detské programovacie prostredia často využívajú rôzne metafory. Napríklad, korytnačka zastupuje (zosobňuje) grafické pero, ktoré kreslí relatívnym pohybom (t.j. nevyužíva súradnice). Vďaka metaforám ľahšie sprístupníme rôzne informatické koncepty, algoritmické konštrukcie, techniky aj mladším deťom.

Aktivita 18:

Porovnajte prostredia Imagine a Scratch:

- Ako sa riadi pohyb hrdinov – korytnačky a mačky?
- Aké výhody a aké nevýhody má zápis programov pomocou kartičiek? Porovnajte ho so zápisom programov v prostredí Imagine.
- Aké programové konštrukcie ste objavili?
- Na aké udalosti dokáže hlavný hrdina reagovať?

Aktivita 19:

Uvažujte a diskutujte nad tým, aká je úloha programovania v škole. Do akej miery má a dokáže škola reagovať na požiadavky softvérového priemyslu?

Zo žiakov neplánujeme vychovať profesionálnych programátorov, vývojárov (veď možno až pre 99% žiakov nebude programovanie povolaním). Našim cieľom nie je perfektne naučiť programovací jazyk, jeho knižnice, vybranú množinou algoritmov, ani špecializované technológie. Samozrejme, ak sa žiaci dožadujú náročnejší tém alebo chcú spoznať určité technológie, budeme radi, ak ich učiteľ zvládne naučiť. Najmä v súvislosti s hodnotením žiakov si však musíme uvedomiť, že toto nie sú povinné zložky školskej informatiky.

Domnievame sa, že výber programovacieho jazyka a vývojového prostredia je veľmi dôležitý pre žiaka – začiatočníka. Dôležitý je aj pre učiteľa. Zlý výber znamená, že sa natrápia nielen žiaci, ale veľmi sa bude trápiť aj učiteľ.

Žiaci budú musieť okrem, pre začiatočníka, náročných algoritmických konceptov, zvládnuť aj komplikácie s jazykom. Žiaci ľahko napíšu zlý kód. Napríklad, v jazyku C++ sa začiatočníci často mýlia a namiesto porovnania:

```
a==1
```

píšu v podmienenom príkaze priradenie:

```
a=1
```

Alebo, namiesto zreťazenia sa sčítajú smerníky:

```
"Vysledok=" + 1000 * 500
```

Napriek tomu kompilátor preloží takýto zlý program, ktorý ale nebude správne pracovať, bude sa správať podivne, bude padať na čudných miestach. Väčšina začiatočníkov nemá šancu takéto chyby nájsť a opraviť ich. Učiteľ im bude musieť pomáhať a vysvetľovať chyby (čo je v istých situáciách problematické).

Aby dokázal učiteľ žiakom pomôcť, bude musieť spoznať veľa rôznych pascí, zlých situácií a nesprávnych kombinácií, ktoré sú často dôsledkom syntaxe jazyka. Tiež si treba uvedomiť, že pri voľbe nevhodného jazyka alebo prostredia nie je jednoduché venovať sa naraz 15 žiakom, zvládnuť hľadanie, opravu a prípadne aj vysvetľovanie chýb. Vyučovanie v žiadnom prípade nemôže vyzerať tak, že v triede je 15 žiakov, učiteľ sa dvom venuje, pretože tí ho ešte vnímajú, prípadne s ním súhlasia a ostatní žiaci netušia, o čom učiteľ rozpráva.

Učebnice, knihy a vzdelávacie materiály

Cieľ: Spoznať vzdelávacie materiály pre základnú a strednú školu, porozumieť tomu, že sú určené pre inú cieľovú skupinu ako odborná literatúra.

Aktivita 20:

Pracujte v tímoch. Každý tím dostane učebnicu. Preštudujte ju a pokúste sa zodpovedať na nasledujúce otázky:

- Aký programovací jazyk sa používa?
- Aké prostredie používajú autori pri vysvetľovaní?
- Aké témy a v akom poradí sa preberajú? Stručne ich vymenujte.
- Aký je váš dojem?

Po preštudovaní zvoľte jedného zástupcu, ktorý v krátkosti prezentuje výsledky svojej skupiny pred ostatnými kolegami.

Priebeh:

- Študenti si pozrú učebnice alebo pracovné listy – zväčša stačí 15 až 20 minút, potom postupne referujú o učebnici svojim spolužiakom.
- Vystúpenia niektorých študentov môžu byť povrchné – napríklad si všímajú počet strán, či je učebnica čiernobiela a podobne. Lektor by sa mal v takých prípadoch vystúpenie usmerňovať a pýtať sa otázky z aktivity, prípadne doplniť niektoré zaujímavé informácie.

Štýl, akým sú písané odborné knihy alebo web stránky o programovaní, je v podstatnej miere odlišný od štýlu, akým sú písané učebnice pre školy. Pre kontrast medzi učebnicou a odbornou knihou zvykneme zaradiť do našich seminárov aj nasledujúcu aktivitu.

Aktivita 21:

Každý dostane odbornú knihu alebo odkaz webovú stránku. Preštudujte ju a pokúste sa zodpovedať na nasledujúce otázky:

- O akom programovacom jazyku je?
- Aké prostredie používajú autori pri vysvetľovaní?
- Aké témy a v akom poradí sa preberajú? Stručne ich vymenujte.
- Aký je váš dojem?
- Viete si predstaviť, ako by ste knihu alebo webovú stránku použili vo svojej praxi? Povedzte.

Pri hodnotení si všímajte aj didaktickú stránku:

- Či sa pri vysvetľovaní nových pojmov používajú známe, už vysvetlené a zadefinované pojmy.
- Či je vysvetlenie primerane jednoduché, názorné, sprevádzané obrázkami alebo ilustráciami.
- Či autor používa motivácie, jednoduché ukážkové príklady, úlohy na precvičenie.
- Či má čitateľ, pre ktorého je kniha alebo webová stránka určená, šancu porozumieť textu.

Väčšina odbornej literatúry a web stránok má nasledovnú štruktúru:

- Píšu o typoch, syntaxi a konštrukciách programovacieho jazyka. Nájdeme v nich mnohostranové pasáže o typoch a o syntaktických pravidlách. Takto však nemôžeme postupovať pri vyučovaní v triede, pretože by sme žiakov unudili a zahltili encyklopedickými faktami.
- Nie zriedka nájdeme také web stránky, ktoré do detailov vysvetľujú jednoduché veci – napríklad obsahujú ilustrovaný popis prostredia alebo postupu pri uložení projektu. Avšak náročné a ťažké programátorské koncepty sú vysvetlené nedostatočne, napríklad „čo je“ a „ako funguje“ premenná, je vysvetlené iba jednou vetou.
- Zameriavajú sa na vysvetľovanie komponentov, knižníc a funkcií. Niekedy začínajú jednoduchým príkladom (napríklad, zobrazenie správy „Hello Word!“), pokračujú vysvetľovaním komponentov, programovacích techník v operačnom systéme alebo sa venujú programovaniu databáz.
- Ak autor uvádza príklady, často sa jedná o textovo alebo matematicky orientované ukážky. Na webe sú často príklady, v ktorých sa využívajú rôzne programátorské triky – také, ktoré začiatočníka skôr zmätú, pomýlia.

Odborná literatúra a webové stránky obsahujú veľa technických detailov a siahodlhé encyklopedické zoznamy s informáciami, ktoré sú pre skúseného programátora veľmi cenné, ale pre začiatočníka, žiaka v škole, úplne zbytočné.

O poznávacom procese

Cieľ: Porozumieť procesu, v ktorom si žiaci budujú nové poznatky. Rešpektovať to, že poznatok sa získava v určitých etapách.

Rozdiel medzi tým, čo učíme a ako tomu žiaci porozumejú Je relatívne, čo považujeme za jednoduché my, učitelia a čo žiaci. Nezriedka sa stáva, že mladý učiteľ je zanietený, nadšený, entuziastický a má sklon učiť všetko, čo sám vie. Často si neuvedomuje, že pojmy a algoritmy, ktoré ovláda, má natrénované, pretože s nimi bežne a dlhé roky pracoval. Preto mu zdajú jednoduché. Za meradlo kvality vyučovania rozhodne nebudeme považovať to, koľko informácií učiteľ porozprával – teda, ako sa učiteľ pred žiakmi predvádza. Naopak, za kvalitné vyučovanie budeme považovať také, ktorému porozumie najviac žiakov.

Aktivita 22:

Pozrite si videá s experimentmi Jean Piageta o etapách vývoja detí.

Priebeh:

- Študenti si pozrú video, napríklad usporiadania Piaget on Piaget, Part 2:

  `https://www.youtube.com/watch?v=Qb4TPj1pxzQ`

- Ak študenti video poznajú, stačí, ak si s lektorom pripomenú, či si pamätajú, o čom bolo.

Aktivita 23:

Diskutujte o tom, v ktorom veku dokážu deti vykonávať aké operácie. Čo si myslíte, akým programátorských konceptom v každom veku porozumejú?

V informatike používame veľa odborných termínov a abstraktných pojmov. V potrebe porozumieť abstraktným pojmom sa vyučovanie informatiky do istej miery podobá matematike. Pri vyučovaní informatiky dávame prednosť takému učeniu, pri ktorom žiaci sami zbierajú vlastné skúsenosti. Teda napríklad aj tým, že žiaci algoritmy prakticky programujú. Oproti matematike tak získavame výhodu v tom, že žiaci si môžu algoritmy priamo vyskúšať, overiť, môžu sa s algoritmami „hrať“. Tým získajú ďalšie cenné skúsenosti. Preto sme si „zakázali“ učiť programovanie na papieri.

Aktivita 24:

Spoznajte etapy poznávacieho procesu:

1. etapa	Motivácia	
	↓	
2. etapa	Zbieranie skúseností	… v matematike: separované modely
	↓	
3. etapa	Zovšeobecňovanie, abstrakcia	… v matematike: univerzálny model
	↓	
4. etapa	Poznatok	
	↓	
5. etapa	Trénovanie poznatku	

Priebeh:

- Uvedené označenia etáp sú samé o sebe veľmi abstraktné. Lektor sa môže pýtať študentov, čo si pod nimi predstavujú, ale aj tak treba na konkrétnom príklade ukázať, čo znamenajú.

Motivácia:

„Chceme veľakrát vypísať text `'Teším sa na prázdniny'`“

„Ako vypíšeme jeden taký pozdrav“

```
print('Teším sa na prázdniny')
```

„Ak by sme ich vypísali 5?“

```
print('Teším sa na prázdniny')
print('Teším sa na prázdniny')
print('Teším sa na prázdniny')
print('Teším sa na prázdniny')
print('Teším sa na prázdniny')
```

„Keby sme chceli 10, je to nešikovné ... potrebujeme nejaký mechanizmus, ktorý príkazy zopakuje.„

Zbieranie skúseností:

Riešenie prezradíme a necháme žiakom vyskúšať:

```
for i in range(5):
    print('Teším sa na prázdniny')
```

Žiaci si musia zvyknúť na nové symboly aj odsadzovanie

„Uprav program takto a spusti ho:“

```
for i in range(5):
    print('Teším sa na prázdniny')
    print('====================')
```

Učia sa, že telo cyklu môže mať viac príkazov – tie postupne vykonajú 5x za sebou.

Až teraz povieme, ako program funguje a pomenujeme jednotlivé časti:

– slovom `for` začína príkaz cyklu,

– `range(5)` určuje počet opakovaní,

– ukážeme, čo je telo cyklu.

„Vyskúšaj, čo sa stane keď program takto upravíš:“

```
for i in range(5):
    print('Teším sa na prázdniny')
print('====================')
```

Ďalej žiaci riešia podobné úlohy, menia počet opakovaní, v tele cyklu kreslia, stále používajú konštantný počet opakovaní

Zovšeobecňovanie, abstrakcia – tým, že žiaci riešia veľa podobných, prípadne veľmi jemne stupňovaných úloh, vyjasňujú si pravidlá a vzťahy medzi tým, čo chcú riešiť, čo napíšu a ako program funguje. O cykle vzniká abstraktná predstava (nie sú v nej konkrétne čísla, príkazy):

```
for i in range(počet opakovaní):
    telo cyklu
```

To je už veľmi **upratený** poznatok. V budúcnosti sa však upravuje (napríklad, premenná cyklu sa nemusí nazývať `i`, alebo cyklus sa dá kombinovať inými programovými konštrukciami atď.) Nasleduje veľa úloh na trénovanie.

Aktivita 25:

Pracujte v tíme a používajte učebnicu. Teraz zamerajte sa iba na jednu kapitolu v učebnici a analyzujte ju z pohľadu poznávacieho procesu. Povedzte:

- Akú motiváciu autor použil?
- Na akých príkladoch alebo situáciách zbierajú žiaci prvé skúsenosti s novou témou?
- K akému poznatku autor žiakov nasmeruje?

- Aké úlohy na trénovanie sa v učebnici vyskytujú?

Aktivita 26:

Diskutujte o tom, či sa tak, ako to autor uviedol v učebnici, dá postupovať aj so žiakmi v triede.

O vyučovaní konceptov

Cieľ: Naučiť rozmýšľať o svojom vyučovaní, trénovať si vyučovanie a vysvetľovanie.

Aktivita 27:

Povedzte, aké témy z programovania, a v akom by ste učili svojich žiakov.

Priebeh:

- Študenti navrhujú rôzne poradia tém, lektor sa ich snaží zaznamenať na tabuľu – väčšinou vznikne strom rôznych možností.
- Akceptujú sa rôzne, aj náhodné poradia tém.

Niektoré spôsoby vyučovania programovania, ktoré zachytíme v strome, sú pre nás zaujímavé a poučné. Preto sa pozrieme na nasledujúce tri prístupy vyučovania programovania:

- Neprogramátorská informatika – rozprávanie sa o algoritmoch, aktivity,
- Tradičný prístup – začíname programovať v konzolovom režime,
- Súčasný prístup – začíname programovať s využitím graficky.

Aktivita 28:

Diskutujte o rôznych názoroch na poradie vyučovania programovania: • najskôr treba učiť vývojové diagramy, • najskôr treba učiť zostavovať slovné algoritmy, až potom pracovať pri počítačoch, • najskôr treba učiť objekty, objektové modelovanie, návrhové vzory • najskôr treba učiť ...

Aktivita 29:

Zvoľte si jeden z programátorských konceptov: • kreslenie, • premenné, • cyklus, • vetvenie, • podprogramy. Predstavte si, že máte žiaka alebo skupinu žiakov, ktorí takému konceptu nerozumejú. Vtedy sa im zvykneme viac venovať a uvedený koncept objasniť. Pripravte si vystúpenie, v ktorom predvediete, ako by ste v takej situácii postupovali. Vaše vystúpenie bude mať dve časti. Prvá časť bude obsahovať analýzu témy: • zoznam predpokladov – čo už žiaci vedia, • cieľ – čo chcete svojich žiakov naučiť, • zoznam pojmov a poznatkov, ktoré sa žiaci naučia. Druhá časť bude ukážka toho, ako by ste postupovali pri ich vyučovaní: • treba dodržať etapy poznávacieho procesu (motivácia, zbieranie skúseností...), • čo a akým spôsobom budete vysvetľovať,

- čo a aké úlohy budú riešiť žiaci samostatne.

Počas vystúpenia môžete simulovať situáciu v triede – predpokladajte, že výučba prebieha v počítačovej učebni, každý žiak sedí sám pri počítači, k dispozícii máte tabuľu.

Priebeh:

- Vystúpenie študenta by malo trvať do 20 minút.
- Lektor aj spolužiaci si všímajú, ako kvalitne je téma spracovaná. Po skončení spoločne zhodnotia vystúpenie.

Všímame si, či študent:

- dodržiava etapy poznávacieho procesu,
- zvolil jednoduché a pre žiakov zrozumiteľné príklady,
- vystúpenie premyslel – ako kreslí na tabuľu, či nerobí chaos,
- vystupuje konzistentne – to, čo hovorí, zodpovedá tomu, čo píše na tabuľu,
- buduje **slovník** – pri vysvetľovaní nových vecí používa iba pojmy, ktoré žiaci poznajú.

Hoci sa zdajú uvedené kritéria jednoduché, pre študentov je veľmi náročné dodržať ich. Preto im často nechávame na spracovanie témy dostatok času – na konci jedného seminára témy rozdelíme, na druhom si pozrieme vystúpenia.

Hodnotenie a klasifikácia

Cieľ: Naučiť študentov hodnotiť žiakov.

Aktivita 30:

Použite krátke 10 minútové testy, ktoré písali žiaci a ohodnoťte ich známkou.

Priebeh:

- Lektor zvolí jednu úlohu, ktorú riešili traja rôzny žiaci. Ideálne je, ak je jedno žiacke riešenie výborné, ale zbytočne komplikované, v riešeniach druhého a tretieho žiaka sú rôzne chyby. Riešenia rozmnoží tak, aby každý študent hodnotil všetky 3 varianty.
- Študenti ohodnotia žiacke riešenia a diskutujú o tom, aké známky žiakom udelili.

Študenti zvyknú hodnotiť rovnaké riešenia veľmi rôzne. Je dôležité diskutovať práve o takomto rozpornom hodnotení a zamyslieť sa nad tým, či je to správne. Hodnotenie žiakov je veľmi citlivá záležitosť. My môžeme študentov viesť aspoň k tomu, aby si stanovili také kritéria, že za ktoré žiaci získavajú body. Na ich základe potom pridelia známku. Nemali by postupovať tak, že za každý chybu dostane žiak o stupeň horšiu známku.

Aktivita 31:

Pozrite sa ešte raz na úlohu, ktorú žiaci riešili. Úlohu vyriešte, riešenie rozdeľte na niekoľko etáp a navrhnite bodovanie každú správne vyriešenú etapu. Stanovte si hranice pre známkovanie a ohodnoťte testy podľa takýchto nových kritérií.