

TEMATISKAIS PLĀNS

4. TEMATS

Kā sistemātiski pārbaudīt un atklūdot risinājumu? (15 stundas)

Pilnveidot prasmi sistemātiski pārbaudīt un atklūdot programmatūras risinājumus, identificējot kļūdas programmā, izmantojot programmēšanas vides piedāvātos atklūdošanas līdzekļus, veidojot testa piemērus un analizējot programmas darbību, lai nodrošinātu tās korektu un stabilu darbību.

Tematā realizētie valsts pamatizglītības standarta sasniedzamie rezultāti

- T.O.2.4.6. Veic programmatūras testēšanu atbilstoši izstrādātajai prasību specifikācijai un projektējumam, analizē testēšanas rezultātus un veic nepieciešamos uzlabojumus programmas darbībā.
- T.O.2.4.3. Izstrādā programmatūru, ievērojot programmatūras izstrādes procesa posmus, dokumentē testēšanas rezultātus un pilnveido programmatūras risinājumu.
- T.O.2.4.10. Izmanto programmēšanas valodas dokumentāciju, kļūdu paziņojumus un palīdzības sistēmu programmas kļūdu identificēšanai un atklūdošanai.
- T.O.2.4.13. Izstrādā programmu, lietojot programmēšanas valodas konstrukcijas, organizē datu ievadi un veic ievaddatu validāciju, lai nodrošinātu programmas korektu darbību.

3. TEMATS. Kā automātiski apstrādāt un attēlot datus?		15 stundas
1	Kļūdas Python vidē un atklādošana	3
2	Kļūdu aspārde un ievaddatu validācija	3
3	Atklādošanas instrumenti	2
4	Zināšanu pārbaude	
5	Testēšana un testēšanas veidi	3
6	Testpiemēri	3
7	Zināšanu pārbaude	

Atbilstība valsts vispārējās vidējās izglītības standartam

Apakštēmas Nr.	Apakštēmas nosaukums	Standarta punkti
1	Kļūdas Python vidē un atklūdošana	2.4.10
2	Kļūdu aspīrāde un ievaddatu validācija	2.4.13
3	Atklūdošanas instrumenti	2.4.10
5	Testēšana un testēšanas veidi	2.4.6
6	Testpiemēri	2.4.6

Sasniedzamie rezultāti

Zināšanas

- Testējot programmatūru, iespējams pārbaudīt, vai tā darbojas korekti un atbilst uzstādītajām prasībām. (T.Li.2.)
- Atklājošana ir būtiska programmatūras izstrādes procesa daļa, kas palīdz identificēt un novērst kļūdas programmā. (T.Li.2.)
- Programmēšanas procesā var rasties dažādi kļūdu veidi (piemēram, sintakses, izpildes un loģiskās kļūdas), kurus iespējams identificēt pēc programmēšanas vides sniegtajiem kļūdu paziņojumiem. (T.Li.2.)
- Programmas kļūdas var rasties arī nepareizi izvēlētu datu tipu vai neparedzētu ievaddatu dēļ. (T.Li.2.)
- Ievaddatu validācija palīdz nodrošināt programmas korektu darbību un novērst kļūdas programmas izpildes laikā. (T.Li.2.)
- Programmas darbību iespējams pārbaudīt, izmantojot iepriekš sagatavotus testa piemērus ar dažādiem ievaddatiem. (T.Li.2.)
- Testējot programmu ar lietotājiem, iespējams identificēt problēmas programmas lietojamībā un noteikt nepieciešamos uzlabojumus. (T.Li.1., T.Li.3.)

Prasmes

- Identificē un analizē kļūdas programmā, izmantojot programmēšanas vides sniegtos kļūdu paziņojumus un programmas izpildes rezultātus. (T.V.1.3.2.)
- Veic programmas atklājošanu, pakāpeniski analizējot programmas darbību un veicot nepieciešamos labojumus kodā. (T.V.1.3.2.)
- Izmanto vienkāršus atklājošanas paņēmienus (piemēram, starprezultātu izvadīšanu vai programmas izpildes soļu analīzi), lai noteiktu kļūdas cēloni. (T.V.1.3.2.)
- Izstrādā testa piemērus un testa komplektus programmas darbības pārbaudei dažādās situācijās. (T.V.1.3.2.)
- Veic programmatūras testēšanu atbilstoši izstrādātajai programmatūras prasību specifikācijai un projektējumam. (T.O.2.4.6.)
- Analizē testēšanas rezultātus un, balstoties uz tiem, veic programmas uzlabojumus vai novērš konstatētās kļūdas. (T.V.1.3.2.)
- Organizē programmas ievadi, veicot ievaddatu validāciju un pārbaudot programmas darbību ar dažādiem ievaddatiem. (T.O.2.4.13.)

Komplekss sasniedzamais rezultāts

- Analizē programmatūras darbību, sistemātiski veicot testēšanu ar dažādiem ievaddatiem, dokumentē testēšanas rezultātus un, balstoties uz iegūtajiem rezultātiem, veic programmas atklājošanu un nepieciešamos uzlabojumus. (T.V.1.3.2., T.O.2.4.6.)

Ierādumi

- Attīsta ieradumu sadarboties ar klasesbiedriem, konstruktīvi apspriežot programmas darbību un korekti norādot uz konstatētajām kļūdām. (Tikumi – laipnība, godīgums; vērtības – cilvēka cieņa, darbs)
- Attīsta ieradumu uztvert kļūdas kā mācīšanās un pilnveides iespēju, analizējot testēšanā atklātās problēmas un cenšoties tās turpmāk novērst. (Tikumi – gudrība, godīgums; vērtība – darbs)
- Attīsta ieradumu sistemātiski pārbaudīt programmas darbību un rūpīgi testēt izstrādāto risinājumu pirms tā nodošanas lietošanai. (Tikums – atbildība; vērtība – darbs)
- Attīsta ieradumu dokumentēt testēšanas rezultātus un veikt programmas uzlabojumus, balstoties uz iegūtajiem rezultātiem. (Tikums – atbildība; vērtība – darbs)

Temata saturs un tā realizācija

Temata apakšdaļa	Saturs
Kļūdas Python vidē un atklāšana	Skolēni apgūst programmēšanas kļūdu atpazīšanas un labošanas pamatprincipus, veidojot izpratni par to, kā analizēt programmas darbību un noteikt problēmu cēloni. Tēmas ietvaros skolēni iepazīstas ar dažādiem programmēšanas kļūdu veidiem, to pazīmēm un kļūdu paziņojumu nozīmi, vienlaikus attīstot prasmi patstāvīgi analizēt programmu darbību un veikt nepieciešamos labojumus. Aktualizācijas posmā skolēni diskutē par to, kāpēc programmēšanas procesā rodas kļūdas un kā tās ietekmē programmas darbību. Skolēni analizē situācijas, kurās programma nedarbojas kā paredzēts, un secina, ka spēja pamanīt, izskaidrot un novērst kļūdas ir būtiska programmētāja darba sastāvdaļa. Apjēgšanas posmā skolēni iepazīstas ar dažādiem programmēšanas kļūdu veidiem un mācās analizēt kļūdu paziņojumus, lai noteiktu problēmas cēloni. Skolēni attīsta izpratni par to, kā programmēšanas valodas vide informē par kļūdām, kā interpretēt paziņojumos norādīto informāciju un kā soli pa soli meklēt problēmas vietu programmā. Tiek veidota izpratne arī par to, ka dažas kļūdas rodas programmas rakstīšanas laikā, savukārt citas parādās tikai programmas izpildes laikā. Praktiskajā darbībā skolēni analizē dotus programmu piemērus un veic nepieciešamos labojumus, balstoties uz kļūdu paziņojumiem un programmas loģikas analīzi. Skolēni trenējas noteikt kļūdas tipu, izskaidrot tās rašanās iemeslu un izvēlēties piemērotu labošanas pieeju. Pakāpeniski skolēni attīsta prasmi patstāvīgi pārbaudīt programmas darbību, analizēt rezultātus un izlabot programmu tā, lai tā darbotos atbilstoši uzdevuma nosacījumiem. Refleksijas posmā skolēni izvērtē, kādas kļūdas bija vieglāk atpazīt un kurās situācijās kļūdu cēloņa noteikšana prasīja dziļāku programmas analīzi. Skolēni secina, ka kļūdu analīze ir svarīga programmēšanas procesa daļa, kas palīdz labāk izprast programmas darbību un attīstīt sistemātisku pieeju problēmu risināšanā. Tādējādi veidojas pamats turpmākai programmēšanas prasmju attīstībai un efektīvai darbībai programmēšanas vidē.
Kļūdu apstrāde un ievaddatu validācija	Skolēni apgūst programmas stabilitātes nodrošināšanas pamatprincipus, veidojot izpratni par kļūdu apstrādes un ievaddatu validācijas nozīmi programmēšanā. Tēmas ietvaros skolēni iepazīstas ar metodēm, kas palīdz novērst negaidītas programmas darbības kļūmes, pārbaudīt lietotāja ievadītos datus un nodrošināt, ka programma spēj korekti darboties arī neparedzētās situācijās. Aktualizācijas posmā skolēni analizē situācijas no ikdienas, kurās nepareizi ievadīti dati var radīt kļūdas vai neparedzētus rezultātus digitālajās sistēmās. Skolēni diskutē par to, kāpēc programmas var nedarboties korekti, ja lietotājs ievada nepareizu informāciju vai ja programma nav sagatavota dažādām situācijām. Tādējādi tiek veidota izpratne par nepieciešamību kontrolēt ievaddatus un apstrādāt kļūdas programmas darbības laikā. Apjēgšanas posmā skolēni iepazīstas ar kļūdu apstrādes principiem un programmēšanas konstrukcijām, kas ļauj kontrolēt programmas darbību gadījumos, kad rodas neparedzētas situācijas. Skolēni apgūst, kā iespējams uztvert programmas izpildes laikā radušās kļūdas un nodrošināt, lai programma turpinātu darbu vai sniegtu lietotājam saprotamu paziņojumu. Vienlaikus skolēni apgūst ievaddatu validācijas principus, analizējot, kā programmā pārbaudīt datu tipu, formātu un vērtību intervālu, pirms šie dati tiek izmantoti aprēķinos vai citās darbībās. Tiek veidota izpratne par tipisku datu apstrādes plūsmu programmā – datu ievade, pārbaude, apstrāde un rezultāta attēlošana.

	Praktiskajā darbībā skolēni izstrādā vienkāršas programmas, kurās tiek izmantota kļūdu apstrāde un ievaddatu validācija. Skolēni papildina programmas ar datu pārbaudes nosacījumiem un mehānismiem, kas ļauj novērst kļūdas, ja lietotājs ievada nepareizu vai negaidītu informāciju. Praktiskie uzdevumi ļauj skolēniem salīdzināt dažādas pieejas datu pārbaudei, izmantot nosacījumu konstrukcijas un teksta apstrādes metodes, kā arī novērtēt, kā šādi risinājumi padara programmu drošāku un lietotājam saprotamāku. Refleksijas posmā skolēni izvērtē, kā kļūdu apstrāde un ievaddatu validācija palīdz uzlabot programmas kvalitāti un uzticamību. Skolēni secina, ka programmā ir svarīgi paredzēt dažādus lietotāja rīcības scenārijus un nodrošināt, ka programma spēj korekti reaģēt arī uz neparedzētiem ievaddatiem. Tādējādi veidojas izpratne par atbildīgu programmēšanas pieeju, kur programmas stabilitāte un lietotāja pieredze ir tikpat svarīga kā programmas funkcionalitāte.
Atklūdošanas instrumenti	Skolēni apgūst atklūdošanas pamatprincipus, veidojot izpratni par to, kā analizēt programmas izpildes gaitu, noteikt kļūdu cēloņus un izmantot koda redaktora atklūdošanas instrumentus programmas labošanai. Tēmas ietvaros skolēni iepazīstas ar dažādām atklūdošanas pieejām, piemēram, kļūdu paziņojumu analīzi, papildu izvada izmantošanu un mainīgo vērtību pārbaudi, kā arī ar atklūdošanas režīma iespējām koda redaktorā. Aktualizācijas posmā skolēni analizē, kā programmēšanas procesā iespējams noskaidrot, kas tieši notiek programmas darbības laikā, un diskutēt par to, kā labāk saprast koda izpildes secību un mainīgo vērtību izmaiņas. Skolēni secina, ka kļūdu novēršanai nepietiek tikai ar gala rezultāta apskati, bet bieži ir nepieciešams izsekot programmas darbību soli pa solim. Apjēgšanas posmā skolēni iepazīstas ar atklūdošanas instrumentu iespējām koda redaktorā. Skolēni apgūst, kā izmantot apstāšanās punktus, kā apturēt programmas darbību noteiktā vietā, kā soļot pa kodu pa vienai darbībai un kā novērot mainīgo pašreizējo stāvokli programmas izpildes laikā. Tiek veidota izpratne par to, ka atklūdošanu iespējams veikt gan ar speciāliem instrumentiem, gan bez tiem, taču atklūdošanas režīms palīdz ātrāk un pārskatāmāk saskatīt kļūdu cēloņus un programmas loģiku. Praktiskajā darbībā skolēni izmanto atklūdošanas instrumentus, lai analizētu un labotu kļūdas dotās programmās. Sākotnēji skolēni seko skolotāja demonstrācijai, kur tiek parādīts, kā palaist atklūdošanas režīmu, izmantot apstāšanās punktus un novērot mainīgo vērtības. Pēc tam skolēni praktiski pielieto šīs prasmes, risinot uzdevumus, kuros nepieciešams analizēt programmas darbību, noteikt kļūdas vietu un veikt nepieciešamos labojumus. Pakāpeniski skolēni nostiprina prasmi patstāvīgi izmantot atklūdošanas instrumentus dažādās situācijās. Refleksijas posmā skolēni izvērtē, kā atklūdošanas instrumenti palīdz labāk izprast programmas darbību un paātrina kļūdu labošanas procesu. Skolēni secina, ka atklūdošana ir nozīmīga programmēšanas darba sastāvdaļa, jo tā palīdz ne tikai atrast kļūdas, bet arī dziļāk saprast programmas loģiku, datu plūsmu un mainīgo stāvokļu maiņu. Tādējādi veidojas pamats sistemātiskai un pārdomātai pieejai programmu analīzē un uzlabošanā.
Testēšana un testēšanas veidi	Skolēni apgūst testēšanas un specifiskācijas saistību programmatūras izstrādē, veidojot izpratni par to, ka kvalitatīva testēšana nav iespējama bez skaidri formulētām prasībām un lietošanas nosacījumiem. Tēmas ietvaros skolēni iepazīstas ar testēšanas nozīmi, dažādiem testēšanas veidiem un specifiskācijas lomu programmatūras kvalitātes nodrošināšanā, vienlaikus attīstot prasmi precizēt prasības tā, lai tās būtu saprotamas un pārbaudāmas. Aktualizācijas posmā skolēni analizē, kāpēc testēšana ir nepieciešama un kādas sekas var rasties, ja programmatūra netiek pietiekami pārbaudīta. Skolēni diskutē par to, ka katram produktam ir paredzētais lietošanas veids, ierobežojumi un nosacījumi, un secina, ka testēšana palīdz pārliecināties, vai risinājums atbilst šīm prasībām un darbojas droši paredzētajās situācijās.

	Apjēgšanas posmā skolēni iepazīstas ar testēšanas lomu programmatūras izstrādes procesā, analizē dažādus testēšanas veidus un saprot, ka katrs no tiem palīdz pārbaudīt risinājumu no cita skatpunkta. Vienlaikus skolēni veido izpratni par specifiskācijas nozīmi, apzinoties, ka testēšana pārbauda programmatūras atbilstību uzstādītajām prasībām. Tiek skaidrots, ka nepilnīga vai neskaidra specifiskācija apgrūtina gan programmas izstrādi, gan tās testēšanu, jo nav iespējams precīzi noteikt, kādam rezultātam jābūt. Praktiskajā darbībā skolēni vispirms analizē dažādu objektu vai risinājumu lietošanas iespējas un ierobežojumus, tādējādi nostiprinot izpratni par prasību formulēšanu un paredzēto lietojumu. Turpinājumā skolēni salīdzina nepilnīgi formulētas un precizētas prasības, kopīgi diskutē par uzlabojumiem un praktiski pilnveido izvēlēta risinājuma specifiskāciju. Darba gaitā skolēni formulē skaidrākus nosacījumus, konkretizē funkcionalitāti un precizē lietošanas scenārijus, lai izveidotā specifiskācija būtu labāk izmantojama programmas izstrādei un testēšanai. Refleksijas posmā skolēni izvērtē, kā specifiskācijas kvalitāte ietekmē testēšanas iespējas un programmatūras izstrādes procesu kopumā. Skolēni secina, ka jo precīzāk aprakstītas prasības, jo vieglāk pārbaudīt programmas atbilstību, savlaicīgi pamanīt nepilnības un izveidot lietotāja vajadzībām atbilstošu risinājumu. Tādējādi veidojas izpratne par testēšanu kā sistemātisku procesu, kas balstās uz skaidru, pārdomātu un pārbaudāmu specifiskāciju.
Testpiemēri	Skolēni apgūst testpiemēru izstrādes pamatprincipus, veidojot izpratni par to, kā strukturēti un atkārtojami pārbaudīt programmatūras darbību. Tēmas ietvaros skolēni iepazīstas ar funkcionālās un nefunkcionālās testēšanas atšķirībām, testpiemēru pieraksta struktūru, kā arī ar pozitīvo, negatīvo un robežgadījumu testu nozīmi programmatūras kvalitātes nodrošināšanā. Aktualizācijas posmā skolēni analizē, kāpēc testēšanai ir nepieciešama skaidra pieraksta struktūra un kāpēc nepietiek tikai ar intuitīvu programmas izmēģināšanu. Skolēni diskutē par to, ka programmatūra var izpildīt paredzētās funkcijas, bet vienlaikus būt neērta, nedroša vai nestabila, tādēļ testēšanā jāņem vērā gan funkcionalitāte, gan citi kvalitātes aspekti. Apjēgšanas posmā skolēni iepazīstas ar testpiemēru tipisko struktūru, kurā ietilpst nosaukums, priekšnosacījumi, testa soļi un paredzētais rezultāts. Skolēni apgūst atšķirību starp funkcionālajiem un nefunkcionālajiem testiem, kā arī izprot, kāpēc ir svarīgi veidot ne tikai pozitīvus, bet arī negatīvus testpiemērus. Tiek veidota izpratne par to, ka kvalitatīvs testpiemērs ļauj citiem atkārtot pārbaudi vienādi un iegūt salīdzināmus rezultātus, savukārt precīza specifiskācija palīdz noteikt, ko un kā tieši nepieciešams pārbaudīt. Praktiskajā darbībā skolēni vispirms trenējas rakstīt strukturētus testpiemērus dažādām jau gatavām digitālām sistēmām, analizējot to funkcijas un paredzēto lietojumu. Turpinājumā skolēni izstrādā negatīvus testpiemērus, kuros tiek pārbaudīta sistēmas reakcija uz nepareiziem vai neparedzētiem datiem. Noslēgumā skolēni praktiski pielieto iegūtās zināšanas konkrētai Python lietotnei, izstrādājot gan pozitīvus, gan negatīvus testpiemērus un īpašu uzmanību pievēršot robežvērtībām un īpašiem ievaddatu gadījumiem. Refleksijas posmā skolēni izvērtē, kuri testpiemēri bija vieglāk izstrādājami un kuri prasīja dziļāku domāšanu par sistēmas darbību un iespējamām kļūmēm. Skolēni secina, ka testpiemēru rakstīšana palīdz ne tikai pārbaudīt programmatūru, bet arī labāk izprast prasības, paredzēt problēmsituācijas un veidot kvalitatīvākus, drošākus un uzticamākus risinājumus. Tādējādi veidojas pamats sistemātiskai testēšanas pieejai, kur pārbaude balstās uz skaidru struktūru, precīziem scenārijiem un pārdomātu rezultātu izvērtēšanu.