

TEMATISKAIS PLĀNS

1. TEMATS

Kā rakstīt programmu, izmantojot izvēlēto programmēšanas valodu? (60 stundas)

Pilnveidot prasmi rakstīt programmas, izmantojot izvēlēto programmēšanas valodu, ievērojot tās sintaksi un pamatkonstrukcijas. Veidot izpratni par datu ievadi, apstrādi un izvadi, lietojot mainīgos, nosacījumu konstrukcijas, ciklus, datu struktūras un funkcijas. Attīstīt prasmi strukturēti veidot un skaidrot programmas darbību, analizējot uzdevuma nosacījumus un pārbaudot izstrādātā risinājuma korektumu.

Tematā realizētie valsts pamatizglītības standarta sasniedzamie rezultāti

- T.O.2.4.1. Analizē vienkāršus ikdienas darba procesus un saskata tajos vai to daļās automatizācijas iespējas.
- T.O.2.4.12. Izmanto pedagoga ieteikto programmēšanas valodu un tai atbilstošu programmatūras izstrādes vidi programmu veidošanai.
- T.O.2.4.13. Lieto programmēšanas valodas pamatkonstrukcijas (mainīgos, ievadi un izvadi, loģiskās izteiksmes, zarošanos, ciklus un funkcijas), organizējot datu ievadi un izvadi.
- T.O.2.4.8. Veido programmas kodu, ievērojot programmēšanas valodas sintaksi, koda strukturēšanu un komentēšanu.
- T.O.2.4.10. Izmanto programmēšanas valodas dokumentāciju un palīdzības sistēmu programmas izstrādes procesā.
- T.O.2.4.14 Izmanto vienkāršas datu struktūras (viendimensiju un divdimensiju masīvus/sarakstus, teksta datnes) un gatavus algoritmus to apstrādei.
- 3.1.1. Ievēro darba vides ergonomikas un drošības pamatprasības, organizējot darbu ar programmvadāmām ierīcēm.
- 3.1.2. Ievēro darba drošības prasības un veic pasākumus drošai darba organizēšanai programmatūras izstrādes procesā.
- 3.1.5. Ievēro programmatūras licences nosacījumus, intelektuālā īpašuma un personas datu aizsardzības prasības, tostarp izmantojot MI rīkus.

1. TEMATS. Kā rakstīt programmu, izmantojot izvēlēto programmēšanas valodu?		60 stundas
1	Ievads programmēšanā un programmēšanas valodas izvēle	2
2	Teksta izvade un formatēšana programmā	2
3	Mainīgo izmantošana un matemātiskie aprēķini programmā	2
4	Lietotāja ievade un datu tipu pārveidošana programmā	2
5	Nosacījumu konstrukcijas un lēmumu pieņemšana programmēšanā	3
6	Zināšanu pārbaude Nr. 1	2
7	Datu struktūras: sarakstu izmantošana	3
8	Cikla konstrukcijas un to pielietošana	6
9	Divdimensiju datu struktūras (saraksti)	6
10	Zināšanu pārbaude Nr. 2	2
11	Funkciju veidošana un izmantošana programmā	4
12	Teksta apstrādes pamatfunkcijas programmēšanā	5
13	Darbs ar teksta datnēm programmēšanā	5
14	Zināšanu pārbaude Nr. 3	2
15	Kontrolēta MI rīku izmantošana programmas pirmkoda izstrādē	6
16	Programmatūras projekta izstrāde (konsoles programma)	5
17	Noslēguma pārbaudījums	3

Atbilstība valsts vispārējās vidējās izglītības standartam

Apakštēmas Nr.	Apakštēmas nosaukums	Standarta punkti
1	Ievads programmēšanā un programmēšanas valodas izvēle	2.4.1, 2.4.12, 3.1.1, 3.1.2, 3.1.5
2	Teksta izvade un formatēšana programmā	2.4.13
3	Mainīgo izmantošana un matemātiskie aprēķini programmā	2.4.13
4	Lietotāja ievade un datu tipu pārveidošana programmā	2.4.13
5	Nosacījumu konstrukcijas un lēmumu pieņemšana programmēšanā	2.4.13
6	Zināšanu pārbaude Nr. 1	
7	Datu struktūras: sarakstu izmantošana	2.4.14
8	Cikla konstrukcijas un to pielietošana	2.4.13
9	Divdimensiju datu struktūras (saraksti)	2.4.14
10	Zināšanu pārbaude Nr. 2	
11	Funkciju veidošana un izmantošana programmā	2.4.13, 2.4.8
12	Teksta apstrādes pamatfunkcijas programmēšanā	2.4.13
13	Darbs ar teksta datnēm programmēšanā	2.4.14
14	Zināšanu pārbaude Nr. 3	
15	Kontrolēta MI rīku izmantošana programmas pirmkoda izstrādē	2.4.10, 3.1.5
16	Programmatūras projekta izstrāde (konsoles programma)	2.4.8, 2.4.10, 2.4.13, 2.4.14, 3.1.2
17	Zināšanu pārbaude Nr.	

Sasniedzamie rezultāti

Ziņas

- Programmēšanas valodas tiek izmantotas algoritmu strukturētai un precīzai aprakstīšanai.
- Katras programmēšanas valodas pamatā ir noteikta sintakse, kas jāievēro, lai programma darbotos korekti.
- Mainīgie, datu tipi, nosacījuma operatori, cikli un funkcijas ir galvenie rīki programmas darbības loģikas veidošanai.
- Programmas kodu iespējams strukturēt un komentēt, lai nodrošinātu tā saprotamību un turpmāku pilnveidošanu.
- Programmēšanas valodas dokumentācija un palīdzības sistēmas atbalsta programmas izstrādes procesu.

Prasmes

- Raksta programmas kodu, izmantojot programmēšanas valodu un ievērojot sintakses noteikumus. *(T.O.2.4.8.)*
- Deklarē un lieto mainīgos, izmanto loģiskās izteiksmes un nosacījumu konstrukcijas. *(T.O.2.4.13.)*
- Izmanto cikla konstrukcijas un funkcijas, lai samazinātu koda dublēšanos un uzlabotu programmas loģiku. *(T.O.2.4.13.)*
- Organizē datu ievadi un izvadi, izmantojot standarta ievades un izvades funkcijas. *(T.O.2.4.13.)*
- Strukturē un komentē kodu, ievērojot programmēšanas valodas sintaksi un labās prakses piemērus. *(T.O.2.4.8.)*
- Izmanto programmēšanas valodas dokumentāciju un izstrādes vides palīdzības rīkus uzdevumu risināšanai. *(T.O.2.4.10.)*
- Izmanto vienkāršas datu struktūras (sarakstus, divdimensiju struktūras, teksta datnes) datu apstrādei programmā. *(T.O.2.4.14.)*

Komplekss sasniedzamais rezultāts

- Izstrādā vienkāršu programmu izvēlētajā programmēšanas valodā, izmantojot mainīgos, nosacījumu konstrukcijas, ciklus, funkcijas un vienkāršas datu struktūras. *(T.O.2.4.1., T.O.2.4.8., T.O.2.4.13., T.O.2.4.14.)*
- Programmas izstrādes procesā analizē uzdevuma nosacījumus, pārbauda risinājuma korektumu un skaidro programmas darbību. *(T.O.2.4.1.)*

Ieradumi

- Attīsta ieradumu rakstīt strukturētu, lasāmu un komentētu programmas kodu. *(Tikums – gudrība; vērtība – darbs)*
- Attīsta ieradumu patstāvīgi meklēt risinājumus, izmantojot programmēšanas valodas dokumentāciju un palīdzības materiālus. *(Tikums – atbildība; vērtība – darbs)*
- Attīsta ieradumu atbildīgi un mērķtiecīgi izmantot digitālos rīkus programmēšanas uzdevumu risināšanā. *(Tikums – atbildība; vērtība – drošība)*

Temata saturs un tā realizācija

Temata apakšdaļa	Saturs
Ievads programmēšanā un programmēšanas valodas izvēle	Iepazīstas ar programmēšanas jēdzienu un programmēšanas valodu nozīmi problēmu risināšanā. Analizē vienkāršus ikdienas procesus un saskata tajos automatizācijas iespējas. Iepazīstas ar programmēšanas valodas izvēles kritērijiem un izstrādes vidi.
Teksta izvade un formatēšana programmā	Skolēni apgūst programmas rezultātu attēlošanas pamatprincipus, izmantojot teksta izvades operatoru <i>print()</i>. Mācību procesā tiek aktualizēta nepieciešamība pēc skaidras, strukturētas un lietotājam saprotamas informācijas attēlošanas programmā. Apjēgšanas posmā skolēni iepazīstas ar teksta formatēšanas iespējām, izmantojot speciālos simbolus (<i>\n</i>, <i>\t</i>) un virkņu atkārtošanu, analizējot, kā šie paņēmieni ietekmē izvades struktūru un vizuālo uztveramību. Praktiskajās darbībās skolēni patstāvīgi veido programmas ar strukturētu un tabulētu izvadi, kombinējot dažādus formatēšanas paņēmienus. Refleksijā skolēni izvērtē, kā formatēta izvade uzlabo programmas lasāmību, estētiku un lietojamību, kā arī secina, kā teksta noformējums palīdz labāk saprast programmas rezultātus.
Mainīgo izmantošana un matemātiskie aprēķini programmā	Skolēni apgūst mainīgo jēdzienu kā pamatu datu glabāšanai un apstrādei programmā, sasaistot to ar iepriekš apgūto teksta izvadi un formatēšanu. Aktualizācijas posmā tiek izmantoti piemēri no matemātikas un ikdienas situācijām, lai pamatotu mainīgo nepieciešamību aprēķinu veikšanā. Apjēgšanas posmā skolēni iepazīstas ar dažādiem datu tiem (<i>int</i>, <i>float</i>) un to nozīmi aprēķinu precizitātē, kā arī apgūst matemātisko darbību pierakstu programmēšanas vidē, ievērojot darbību secību un formulu loģiku. Praktiskajās darbībās skolēni izmanto mainīgos un matemātiskās izteiksmes reālu uzdevumu risināšanai, tostarp pieslēdz matemātisko funkciju bibliotēku (<i>math</i>) un lieto tās funkcijas formulu realizācijai programmā. Refleksijā skolēni analizē aprēķinu rezultātus, izvērtē mainīgo nosaukumu nozīmi programmas saprotamībā un secina, kā pareizi pierakstītas formulas nodrošina korektu un pārskatāmu programmas darbību.
Lietotāja ievade un datu tipu pārveidošana programmā	Skolēni apgūst lietotāja un programmas mijiedarbības principu, izmantojot datu ievadi ar funkciju <i>input()</i>, sasaistot to ar iepriekš apgūtajām zināšanām par mainīgajiem un teksta izvadi. Aktualizācijas posmā skolēni analizē, kā programma saņem informāciju no lietotāja un kā ievadītie dati tiek saglabāti mainīgajos. Apjēgšanas posmā skolēni iepazīstas ar datu tipu nozīmi (<i>str</i>, <i>int</i>, <i>float</i>) un apgūst ievadīto datu pārveidošanu skaitliskās vērtībās, kā arī formatētas izvades veidošanu, izmantojot f-string un noapaļošanas funkciju <i>round()</i>. Praktiskajās darbībās skolēni realizē programmas, kurās lietotājs ievada datus, tiek veikti aprēķini pēc dotas formulas un rezultāti tiek strukturēti un saprotami attēloti. Tiek pievērsta uzmanība ievades kļūdām un to apstrādei, analizējot tipiskās kļūdas (<i>ValueError</i>) un izmantojot vienkāršu kļūdu apstrādi ar <i>try/except</i>.

	Refleksijas posmā skolēni izvērtē, kā datu tipu pārveidošana ietekmē programmas korektumu un kā pareizi formatēta izvade uzlabo rezultātu uztveramību lietotājam.
Nosacījumu konstrukcijas un lēmumu pieņemšana programmēšanā	Skolēni apgūst nosacījumu izmantošanas principu programmēšanā, sasaistot matemātiskos aprēķinus ar salīdzināšanas un loģiskajām izteiksmēm, lai programma spētu pieņemt lēmumus dažādās situācijās. Tiek nostiprināta izpratne par to, ka nosacījuma izteiksmes rezultāts (<i>True</i> vai <i>False</i>) nosaka programmas turpmāko darbību, un ka lēmumu pieņemšana programmā balstās uz precīzi formulētiem nosacījumiem. Aktualizācijas posmā skolēni analizē ikdienas situācijas, kurās tiek pieņemti lēmumi atkarībā no nosacījumiem, un sasaista tās ar programmēšanā izmantotajiem salīdzināšanas operatoriem. Skolēni atkārto matemātiskos operatorus un pārrunā, kā iegūtie aprēķinu rezultāti var tikt salīdzināti, lai noteiktu dažādas rīcības iespējas programmā. Apjēgšanas posmā skolēni iepazīstas ar salīdzināšanas operatoriem (<code>==</code>, <code>!=</code>, <code><</code>, <code>></code>, <code><=</code>, <code>>=</code>) un loģiskajiem operatoriem (<i>and</i>, <i>or</i>, <i>not</i>), kā arī apgūst nosacījumu struktūru <code>if</code>, <code>elif</code>, <code>else</code> izmantošanu Python vidē. Tiek uzsvērta atkāpju nozīme koda struktūrā un nosacījumu secīga izpilde. Skolēni analizē piemērus, kuros programma, balstoties uz ievadītajiem datiem, izvēlas dažādus izpildes ceļus un izvada atšķirīgus rezultātus. Praktiskajās darbībās skolēni realizē programmas, kurās tiek izmantota lietotāja ievade, veikti aprēķini un, izmantojot nosacījumus, pieņemti lēmumi par rezultātu interpretāciju. Skolēni veido risinājumus, kuros salīdzina skaitliskas vērtības, klasificē rezultātus un noformē izvadi lietotājam saprotamā veidā. Tiek pievērsta uzmanība loģiski korektai nosacījumu formulēšanai, koda lasāmībai un rezultātu pārbaudei ar dažādiem testdatiem. Refleksijas posmā skolēni izvērtē, kā pareizi izveidoti nosacījumi ietekmē programmas darbību, kā loģisko operatoru kombinēšana ļauj realizēt sarežģītākus lēmumu pieņemšanas scenārijus un kā strukturēta izvade palīdz lietotājam interpretēt iegūtos rezultātus. Skolēni formulē secinājumus par nosacījumu nozīmi programmēšanā un to pielietojumu reālu situāciju modelēšanā.
Datu struktūras: sarakstu izmantošana	Skolēni apgūst datu struktūru – sarakstu (<i>list</i>) – izmantošanas principus Python programmēšanas valodā, lai vienā mainīgajā strukturēti uzglabātu un apstrādātu vairākas vērtības. Tiek veidota izpratne par sarakstu kā elastīgu datu struktūru, kas ļauj strādāt ar saistītiem datiem, piemēram, vārdu, mērījumu vai citu secīgu vērtību kopām, sasaistot jauno tematu ar iepriekš apgūtajām zināšanām par mainīgajiem, datu tiem un teksta izvadi. Aktualizācijas posmā skolēni analizē ikdienas situācijas, kurās nepieciešams uzglabāt vairākas līdzīga tipa vērtības, un pārrunā, kā šādas datu kopas līdz šim būtu bijis neērti apstrādāt, izmantojot tikai atsevišķus mainīgos. Skolēni sasaista šos piemērus ar programmēšanas vajadzībām un nonāk pie secinājuma, ka saraksti ļauj datus glabāt strukturēti un pārskatāmi. Apjēgšanas posmā skolēni iepazīstas ar saraksta jēdzienu, tā pierakstu Python vidē un pamata darbībām ar sarakstiem. Tiek apgūta saraksta izveide, elementu pievienošana ar metodi <i>append()</i>, kā arī piekļuve atsevišķiem elementiem, izmantojot indeksus, uzsverot, ka indeksēšana sākas ar 0. Skolēni analizē piemērus, kuros saraksti tiek izmantoti datu ievadei, saglabāšanai un elementārai apstrādei, kā arī iepazīstas ar tipiskākajām kļūdām, kas var rasties, strādājot ar indeksiem. Praktiskajās darbībās skolēni realizē programmas, kurās lietotājs ievada vairākas vērtības, kas tiek saglabātas sarakstā un tālāk apstrādātas. Skolēni trenējas nolasīt atsevišķus saraksta elementus, aizstāt kļūdainas vērtības, veikt vienkāršus aprēķinus ar sarakstā glabātajiem datiem un sagatavot

	strukturētu izvadi. Tiek uzsvērtā datu korektuma pārbaude, saprotamu mainīgo nosaukumu lietošana un izvades noformēšana, lai rezultāts būtu viegli uztverams lietotājam. Refleksijas posmā skolēni izvērtē, kā sarakstu izmantošana atvieglo darbu ar vairākiem datiem, kāpēc pareiza indeksu lietošana ir būtiska programmas korektai darbībai un kā strukturēta datu izvide uzlabo rezultātu interpretāciju. Skolēni formulē secinājumus par sarakstu nozīmi turpmākajos programmēšanas uzdevumos un to izmantošanu reālu datu analīzē.
Cikla konstrukcijas un to pielietošana	Skolēni apgūst atkārtojuma jēdzienu programmēšanā un cikla konstrukciju nozīmi atkārtotu darbību automatizācijā. Tiek veidota izpratne par to, kā cikli ļauj izvairīties no vienāda koda vairākkārtējas rakstīšanas un kā tos izmantot praktisku uzdevumu risināšanā, sasaistot jauno tematu ar iepriekš apgūtajām zināšanām par mainīgajiem, datu struktūrām un nosacījumu konstrukcijām. Aktualizācijas posmā skolēni analizē ikdienas situācijas, kurās darbības tiek atkārtotas noteiktu reižu skaitu vai līdz brīdim, kamēr ir izpildīts kāds nosacījums. Skolēni pārrunā, kā šādas situācijas var modelēt programmā, un secina, ka cikli nodrošina elastīgu un efektīvu atkārtojuma realizāciju. Apjēgšanas posmā skolēni iepazīstas ar cikla <i>for</i> darbības principiem un funkcijas <i>range()</i> izmantošanu atkārtojumu skaita noteikšanai. Tiek analizēti dažādi <i>range()</i> lietošanas veidi, izmantojot sākuma, beigu un soļa parametrus, kā arī skaidrota cikla izpildes secība. Skolēni apgūst cikla mainīgā izmantošanu aprēķinos un rezultātu strukturētā izvadē. Papildus tam skolēni iepazīstas ar ciklu <i>while</i>, izprotot tā darbību kā atkārtojumu, kas turpinās, kamēr ir spēkā noteikts nosacījums. Praktiskajās darbībās skolēni realizē programmas, kurās cikli tiek izmantoti datu apstrādei, aprēķinu veikšanai un ievades validācijai. Skolēni izmanto ciklu <i>for</i>, lai apstrādātu datu kopas un realizētu secīgus aprēķinus, kā arī ciklu <i>while</i>, lai atkārtotu darbības līdz noteikta nosacījuma izpildei. Tiek pievērsta uzmanība cikla vadībai, izmantojot <i>break</i> un <i>continue</i>, kā arī tipiskajām kļūdām, kas saistītas ar bezgalīgiem cikliem vai nepareizi formulētiem nosacījumiem. Kā algoritmiskās domāšanas piemērs tiek analizēts un realizēts uzdevums FizzBuzz, kas apvieno ciklus ar nosacījumu loģiku. Refleksijas posmā skolēni izvērtē atšķirības starp cikliem <i>for</i> un <i>while</i>, analizē, kādos gadījumos katrs no tiem ir piemērotāks, un secina, kā ciklu izmantošana uzlabo programmas struktūru, lasāmību un efektivitāti. Skolēni formulē secinājumus par ciklu nozīmi turpmākajos programmēšanas uzdevumos un sarežģītāku algoritmu izstrādē.
Divdimensiju datu struktūras (saraksti)	Skolēni apgūst divdimensiju datu struktūru jēdzienu, veidojot izpratni par to, kā dati var tikt organizēti režģa vai tabulas veidā. Temata apguve balstās uz iepriekš apgūtajām zināšanām par sarakstiem, indeksiem un cikliem, paplašinot tās līdz situācijām, kur vienā sarakstā tiek glabāti citi saraksti ("saraksts sarakstā"). Skolēni iemācās sasaistīt 2D saraksta struktūras izmantojot ar reālām dzīves situācijām saistītus uzdevumus (sēdvietu plāns, tabulas, spēļu laukumi) un saprot, kā šādas struktūras izmanto programmēšanā, lai aprakstītu sistēmu stāvokli. Aktualizācijas posmā skolēni analizē ikdienas piemērus, kur informācija ir izvietota rindās un kolonnās, un secina, ka šādai informācijas struktūrai programmēšanā ir nepieciešams atbilstošs datu modelis. Skolēni salīdzina vienas dimensijas sarakstu ar divdimensiju sarakstu un formulē atšķirības: vienas dimensijas saraksts glabā vērtību rindu, bet divdimensiju saraksts glabā rindas, kur katra rinda satur vairākas vērtības (kolonnas). Apjēgšanas posmā skolēni apgūst divdimensiju saraksta izveidi un piekļuvi elementiem, izmantojot pierakstu <i>saraksts[rinda][kolonna]</i>. Tiek skaidrota indeksu nozīme un tas, ka numurēšana sākas ar 0, kā arī tiek nostiprināta prasme noteikt, vai konkrēta vieta režģī ir brīva vai aizpildīta

	(piemēram, ar "-", 0, "X"). Skolēni praktiski izmēģina darbības ar atsevišķām šūnām, ar rindām un kolonnām, veidojot ieradumu vienmēr pārbaudīt indeksu korektumu un datu stāvokli pirms izmaiņu veikšanas. Praktiskajās darbībās skolēni risina uzdevumus, kuros 2D saraksts tiek izmantots datu galbāšanai konkrētam situācijas modelim. Skolēni aizpilda laukumu ar vērtībām, atjaunina režģa šūnas pēc izvēlētas rindas un kolonnas, kā arī veic vienkāršu rindas vai kolonnas izvadi, nostiprinot prasmi izmantot for ciklu datu attēlošanai. Tālāk skolēni apgūst ligzdotu ciklu (<i>nested loop</i>) izmantošanu, lai apstrādātu un izvadītu visu 2D sarakstu tabulas veidā: ārējais cikls organizē rindu apstrādi, bet iekšējais cikls apstrādā kolonnas konkrētajā rindā. Skolēni veido izpratni par ligzdotu ciklu izpildes secību un to, kā tā nodrošina pilnu režģa “pārslasi”. Temata noslēdzošajā posmā skolēni pielieto apgūtās prasmes praktiskā mini-projektā ar spēles laukuma modeli (3 × 3 režģis), kur 2D saraksts apraksta spēles stāvokli. Skolēni realizē lietotāja ievadi (rinda/kolonna), pārbauda, vai izvēlēta vieta ir brīva, atjaunina laukumu ar atzīmi un izvada laukumu uz ekrāna tabulas veidā, izmantojot ligzdotus ciklus. Uzsvars tiek likts uz korektu indeksa lietojumu, stāvokļa pārbaudi pirms gājiena un skaidru vizuālu rezultāta attēlojumu, kas ļauj lietotājam saprast programmas darbību. Refleksijas posmā skolēni izvērtē, kā divdimensiju saraksts palīdz aprakstīt tabulas un spēļu laukumus, un skaidro atšķirību starp piekļuvi rindai (laukums[i]) un piekļuvi konkrētai šūnai (laukums[i][j]). Skolēni analizē, kas praktiskajos uzdevumos bija sarežģītākais (ievade, pārbaude vai izvade), un formulē idejas, kā programmu varētu attīstīt tālāk (piemēram, uzvaras noteikšana, gājienu skaits, kļūdu apstrāde, spēles beigu nosacījumi).
Funkciju veidošana un izmantošana programmā	Skolēni apgūst funkciju nozīmi programmēšanā, veidojot izpratni par to, kā savas funkcijas palīdz strukturēt kodu, samazināt dublēšanos un padarīt programmu pārskatāmu un viegli paplašināmu. Tiek veidota saikne ar iepriekš apgūtajām zināšanām par mainīgajiem, ievadi/izvadi, nosacījumu konstrukcijām, cikliem un datu struktūrām, lai skolēni saprastu, ka lielāku uzdevumu var sadalīt vairākās mazākās, skaidri definētās darbībās. Aktualizācijas posmā skolēni analizē ikdienas situācijas, kurās viena darbība tiek veikta pēc vienas un tās pašas shēmas (piemēram, instrukciju soļi, vienādi aprēķini dažādiem datiem, atkārtotas darbības dažādās situācijās). Skolēni pārrunā, kā programmā var “nosaukt” šādu darbību un izsaukt to atkārtoti, secinot, ka funkcija ir veids, kā vienreiz uzrakstītu risinājumu izmantot vairākas reizes. Apjēgšanas posmā skolēni iepazīstas ar funkcijas definēšanu Python valodā, izmantojot def, un atšķirību starp funkcijas definēšanu un izsaukšanu. Skolēni apgūst funkcijas bez parametriem un funkcijas ar parametriem, izprotot, ka parametri ļauj funkciju pielāgot dažādiem ievaddatiem. Tiek skaidrota return nozīme un tas, kā funkcija var atgriezt vērtību turpmākai izmantošanai programmā. Skolēni nostiprina izpratni par lokālajiem mainīgajiem funkcijas ietvaros un par to, ka funkcija var gan izvadīt rezultātu, gan atgriezt to aprēķinu veikšanai citur programmā. Praktiskajās darbībās skolēni veido un izmanto funkcijas dažādu uzdevumu risināšanai, piemēram, ģeometrisku aprēķinu veikšanai (laukums, perimetrs, tilpums), atkārtoti izsaucot funkcijas ar dažādiem parametriem un pārbaudot rezultātu korektumu ar testdatiem. Skolēni izstrādā funkcijas arī datu apstrādei, izmantojot ciklus funkcijas iekšienē, piemēram, saraksta elementu summēšanai, lielākās vērtības noteikšanai vai divdimensiju saraksta pārskatāmai izvadišanai. Tiek pievērsta uzmanība tam, kā ligzdotie cikli funkcijās ļauj apstrādāt tabulas tipa datus, saglabājot kodu skaidru un strukturētu. Temata noslēgumā skolēni pielieto funkcijas mini-projektā, kur programmas loģika tiek organizēta kā izvēlņu sistēma ar atsevišķām funkcijām izvēlnes parādīšanai, darbību izpildei un ievades apstrādei. Refleksijas posmā skolēni izvērtē, kā funkciju izmantošana ietekmē programmas lasāmību, uzturējamību un iespējas to paplašināt. Skolēni analizē, kādos gadījumos ir lietderīgi izmantot funkcijas ar parametriem un return, kā arī formulē secinājumus par to, kā funkcijas palīdz veidot kvalitatīvākus un uzticamākus programmu risinājumus turpmākajos, sarežģītākos programmēšanas uzdevumos.

Teksta apstrādes pamatfunkcijas programmēšanā	Skolēni apgūst teksta apstrādes pamatfunkcijas programmēšanā, veidojot izpratni par to, kā teksts tiek apstrādāts, pārveidots un analizēts, izmantojot Python iebūvētās metodes. Tiek nostiprināta saikne ar iepriekš apgūtajām zināšanām par lietotāja ievadi, mainīgajiem, nosacījumiem, cikliem un funkcijām, uzsverot, ka teksta apstrāde ir būtiska daļa praktisku programmu izstrādē. Aktualizācijas posmā skolēni analizē situācijas, kurās lietotāja ievadītais teksts nav vienots vai korekts (atšķirīgs burtu reģistrs, liekas atstarpes, nekonsekventa rakstība). Skolēni pārrunā, kāpēc programmā nepieciešams tekstu normalizēt un sagatavot tālākai apstrādei, lai nodrošinātu korektu salīdzināšanu un apstrādi. Apjēgšanas posmā skolēni iepazīstas ar biežāk lietotajām teksta apstrādes metodēm Python valodā, piemēram, <code>upper()</code>, <code>lower()</code>, <code>capitalize()</code> un <code>strip()</code>, izprotot, ka šīs metodes neatjaunina sākotnējo tekstu, bet atgriež jaunu vērtību. Tiek aplūkota teksta sadalīšana un apvienošana, izmantojot <code>split()</code> un <code>join()</code>, kā arī teksta daļu aizvietošana ar <code>replace()</code>. Papildus skolēni apgūst teksta analīzes pamatus, izmantojot <code>in</code>, <code>count()</code> un ciklus, lai pārbaudītu simbolu vai vārdu klātbūtni, noteiktu to biežumu un analizētu teksta saturu. Praktiskajās darbībās skolēni realizē uzdevumus, kuros teksts tiek normalizēts, sadalīts vārdos, pārveidots un analizēts atbilstoši dotajiem nosacījumiem. Skolēni izstrādā vienkāršus algoritmus burtu, ciparu un vārdu skaita noteikšanai, izmantojot teksta metodes kombinācijā ar cikliem un nosacījumiem, kā arī strukturē risinājumu, izmantojot atsevišķas funkcijas teksta apstrādes darbībām. Refleksijas posmā skolēni izvērtē, kuras teksta apstrādes metodes ir visnoderīgākās dažādos uzdevumos, analizē atšķirības starp darbu ar tekstu un darbu ar sarakstiem, kā arī secina, kā teksta apstrādes funkciju izmantošana uzlabo programmas kvalitāti, precizitāti un iespējas to paplašināt turpmākajos programmēšanas uzdevumos.
Darbs ar teksta datnēm programmēšanā	Skolēni apgūst darbu ar teksta datnēm programmēšanā, veidojot izpratni par to, kā programma var saglabāt datus failā un vēlāk tos nolasīt atkārtotai izmantošanai. Tiek nostiprināta saikne ar iepriekš apgūtajām zināšanām par mainīgajiem, teksta apstrādes metodēm, cikliem un funkcijām, uzsverot, ka darbs ar datnēm ir būtisks solis no “programmas, kas strādā tikai izpildes brīdī” uz “programmu, kas uzkrāj un apstrādā datus”. Aktualizācijas posmā skolēni analizē ikdienas situācijas, kurās informācija tiek saglabāta un atjaunota (piezīmes, saraksti, žurnāli, rezultātu saglabāšana). Skolēni pārrunā, kāpēc programmā ir nepieciešams saglabāt datus ārpus mainīgajiem, un secina, ka teksta datnes ļauj vienkārši uzkrāt informāciju un turpināt darbu arī pēc programmas aizvēršanas. Apjēgšanas posmā skolēni iepazīstas ar teksta datņu atvēršanas, nolasīšanas un saglabāšanas pamatprincipiem, izmantojot <code>open()</code> un faila režīmus ("<code>r</code>", "<code>w</code>", "<code>a</code>"). Tiek skaidrota atšķirība starp <code>read()</code>, <code>readline()</code> un <code>readlines()</code>, kā arī pareiza darba organizēšana ar <code>with open(...)</code> as f:. Skolēni apgūst, kā apvienot nolasīto tekstu ar iepriekš apgūtajām teksta apstrādes metodēm (<code>strip()</code>, <code>split()</code>, <code>replace()</code>), lai attīrītu rindas, sadalītu datus un sagatavotu tos turpmākai analīzei. Papildus tiek pievērsta uzmanība biežākajām problēmām, piemēram, nepareizai faila atrašanās vietai vai nosaukumam, kā arī teksta kodējumam, ja datnē ir latviešu burti. Praktiskajās darbībās skolēni izstrādā programmas, kas saglabā lietotāja ievadi teksta datnē, papildina datni ar jaunām rindām un nolasot apstrādā datnes saturu. Skolēni izmanto ciklus, lai apstrādātu katru datnes rindu, veido vienkāršus kopsavilkumus (piemēram, rindu skaits, vārdu skaits, noteiktu simbolu vai vārdu biežums), kā arī strukturē risinājumu, izdalot atsevišķas darbības funkcijās (nolasīšana, apstrāde, saglabāšana). Tiek pievērsta uzmanība datu korektai formatēšanai, lai saglabātais saturs būtu pārskatāms un atkārtoti izmantojams.

	Refleksijas posmā skolēni izvērtē, kā datnes izmantošana maina programmas pielietojumu un iespējas, analizē atšķirību starp datu glabāšanu mainīgajos un datu glabāšanu failā, kā arī formulē secinājumus par to, kā teksta datņu apstrāde palīdz risināt reālus uzdevumus (datu uzkrāšana, pārskatīšana, atkārtota analīze) un sagatavo pamatu darbam ar strukturētākiem formātiem turpmākajās tēmās.
Kontrolēta MI rīku izmantošana programmas pirmkoda izstrādē	Skolēni apgūst kontrolētu mākslīgā intelekta izmantošanu programmas pirmkoda izstrādē, veidojot izpratni par MI kā palīgriķu, kura rezultāti vienmēr ir jāpārbauda, jātestē un jāpielāgo. Tiek stiprināta prasme formulēt uzdevumu tā, lai gan cilvēks, gan MI varētu izstrādāt vienoziģu risinājumu, un tiek aktualizēta atbildīga rīcība digitālajā vidē, īpaši saistībā ar pirmkoda kvalitāti, drošību un autorību. Aktualizācijas posmā skolēni analizē MI ģenerētus programmēšanas piemērus, identificē iespējamās kļūdas vai nepilnības un pārrunā, kā šādi risinājumi var ietekmēt programmas darbību. Apjēgšanas posmā skolēni iepazīstas ar ģeneratīvā MI darbības pamatprincipu un tipiskajiem riskiem: halucinācijām un radošuma risku. Tiek apgūti praktiski principi uzvedņu veidošanai, lai ierobežotu MI interpretāciju: precīzi definēts mērķis, skaidras prasības, aizliegumi, kā arī rezultāta formāts. Skolēni nostiprina izpratni par to, ka uzvedne ir uzdevuma formulējums, un jo precīzāks ir uzdevuma apraksts, jo paredzamāks ir rezultāts. Praktiskajā darbā skolēni izvēlas vienkāršu programmas ideju un apraksta tās ievadi un izvadi. Pēc tam skolēni pārveido ideju par strukturētu algoritma aprakstu, piemēram, darbību secību vai pseidokodu, un izmanto šo aprakstu kā uzvedni MI rīkam koda ģenerēšanai. Skolēni pārbauda iegūtā risinājuma darbību un salīdzina MI ģenerēto kodu ar klasesbiedra manuāli izstrādātu risinājumu, analizējot atšķirības starp tiem un iespējamās iemeslus. Noslēgumā skolēni formulē secinājumus par algoritma apraksta precizitātes nozīmi un par nepieciešamību kritiski izvērtēt MI ģenerētus programmēšanas risinājumus. Refleksijas posmā skolēni izvērtē, kādi faktori nodrošina kvalitatīvu MI ģenerētu programmēšanas risinājumu, uzsverot precīza uzdevuma formulējuma, skaidru ierobežojumu un sistemātiskas testēšanas nozīmi. Skolēni formulē principus MI izmantošanai programmēšanas uzdevumos, nosakot, kā strukturēt uzdevuma aprakstu, kā pārbaudīt iegūtā koda darbību un kā izvērtēt, vai piedāvātais risinājums ir atbilstošs uzdevuma prasībām.
Programmatūras izstrādes projekts	Skolēni īsteno programmatūras izstrādes projektu, kura mērķis ir praktiski nostiprināt iepriekš apgūtās Python programmēšanas pamatprasmes, izstrādājot funkcionējošu konsoles lietotni ar izvēlni, datu saglabāšanu failā un vienkāršu datu apstrādi. Projekta gaitā skolēni pakāpeniski pilnveido programmu, ievērojot strukturētu izstrādes procesu un veidojot izpratni par koda organizēšanu, testēšanu un uzlabošanu. Aktualizācijas posmā skolēni sasaista iepriekš apgūtās zināšanas par datu ievadi un izvadi, nosacījumiem, cikliem un funkcijām ar vienota programmēšanas risinājuma izstrādi. Skolēni analizē, kā šie elementi kopā veido lietotājam saprotamu un funkcionālu programmu. Apjēgšanas posmā skolēni iepazīstas ar vienkāršas konsoles programmas struktūru, analizējot izvēlnes vadības ciklu, datu saglabāšanu un nolasīšanu no faila, kā arī datu apstrādi ar funkciju palīdzību. Skolēni veido izpratni par datu formāta konsekvensi, ievades pārbaudes nozīmi un tipiskām kļūdām, kas rodas darbā ar lietotāja ievadi un failiem. Praktiskajās darbībās skolēni pāros izstrādā programmas funkcionalitāti, pakāpeniski realizējot ierakstu pievienošanu, datu attēlošanu, kopsavilkuma aprēķinus, meklēšanu un filtrēšanu, kā arī programmas testēšanu. Darba procesā skolēni sadala pienākumus, pārbauda viens otra risinājumus un veic nepieciešamos labojumus.

	Refleksijas posmā skolēni izvērtē projekta izstrādes procesu, analizē biežāk pieļautās kļūdas un secina, kā strukturēta izstrāde, testēšana un sadarbība palīdz uzlabot programmas kvalitāti. Skolēni formulē secinājumus par iegūtajām prasmēm un to nozīmi turpmākos programmēšanas uzdevumos.
--	--