doi: 10.15330/msuc.2024.31.52-63

Леся ХРУЩ,
кандидат економічних наук, доцент, доцент кафедри математики та інформатики і методики навчання,
Прикарпатський національний університет імені Василя Стефаника (м. Івано-Франківськ, Україна)

Lesia KHRUSHCH,
Ph. D (Economic Sciences), Associate Professor, Department of Mathematics and Informatics and Methods of Teaching,
Vasyl Stefayk Precarpathian National University (Ivano-Frankivsk, Ukraine)
lesya.khrushch@pu.if.ua
ORCID ID 0000-0002-8559-8712

Оксана ГАРПУЛЬ,
кандидат фізико-математичних наук, доцент,
доцент кафедри математики та інформатики і методики навчання,
Прикарпатський національний університет імені Василя Стефаника (м. Івано-Франківськ, Україна)

Oksana HARPUL,
PhD in Physical and Mathematical Sciences, Associate Professor,
Department of Mathematics and Informatics and Methods of Teaching,
Vasyl Stefanyk Precarpathian National University (Ivano-Frankivsk, Ukraine)
oksana.harpul@pnu.edu.ua
ORCID ID 0000-0002-7441-1369

Володимир ПИЛИПІВ,
доктор фізико-математичних наук, професор кафедри математики та інформатики і методики навчання,
Прикарпатський національний університет імені Василя Стефаника (м. Івано-Франківськ, Україна)

Volodymyr PYLYPIV,
Doctor of Physical and Mathematical Sciences, Professor,
Department of Mathematics and Informatics and Methods of Teaching,
Vasyl Stefanyk Precarpathian National University (Ivano-Frankivsk, Ukraine)
volodymyr.pylypiv@pnu.edu.ua
ORCID ID 0000-0003-0579-1211

УДК 37.018.43:004.4-37.016.6:004.72

ЗАДАЧНИЙ ПІДХІД ДО НАВЧАННЯ ПРОГРАМУВАННЯ В ШКОЛІ: МЕТОДИЧНІ ПІДХОДИ ТА ПРАКТИЧНА РЕАЛІЗАЦІЯ

Анотація. У статті проаналізовано задачний підхід до навчання програмування в школі. Розглянуто методичні підходи, що сприяють розвитку алгоритмічного мислення та практичних навичок учнів. Окреслено шляхи реалізації цього підходу із застосуванням сучасних цифрових інструментів. Особливу увагу приділено адаптації навчального матеріалу відповідно до рівня підготовки школярів.

Визначено актуальність дослідження, що полягає у необхідності добору відповідних завдань і середовищ програмування для кожного учня. Охарактеризовано основні підходи до вивчення програмування в школі: задачний, диференційований, діяльнісний та компетентнісний.

Проаналізовано дослідження щодо задачного підходу та його значення у навчанні програмування. Встановлено, що він сприяє розвитку алгоритмічного мислення, формуванню вмінь моделювати інформаційні процеси, аналізувати умови задачі, будувати алгоритми її розв'язання та реалізовувати їх мовами програмування. Наголошено на необхідності переходу від репродуктивного до творчого програмування.

Зазначено, що головною проблемою навчання програмування є відсутність системного підходу. Підкреслено важливість вивчення не лише синтаксису мов програмування, а й застосування знань для розв'язання реальних задач. Наведено приклад реалізації задачного підходу для учнів 8–11 класів на основі мови Lazarus із використанням блок-схем, алгоритмічних конструкцій і середовищ із візуальним відображенням виконання програм.

Розглянуто методи розв'язування задач у програмуванні, що включають побудову інформаційної моделі, розробку алгоритму та його реалізацію. Визначено етапи розвитку учнівських навичок – від репродуктивного до творчого рівня. Обґрунтовано необхідність диференційованого підходу для максимального розкриття потенціалу учнів у навчанні програмування.

Ключові слова: методичні підходи, алгоритмізація, програмування, задачний підхід, Lazarus, Scratch, педагогічні інновації, інтерактивні методи, ІКТ- компетентності.

THE TASK-BASED APPROACH TO TEACHING PROGRAMMING IN SCHOOLS: METHODOLOGICAL FRAMEWORK AND PRACTICAL IMPLEMENTATION

Abstract. The article analyzes the task-based approach to teaching programming in schools. The study examines methodological approaches that contribute to the development of algorithmic thinking and practical skills among students. The paper outlines ways to implement this approach using modern digital tools, with particular attention given to adapting educational materials based on students' proficiency levels.

The relevance of the study lies in the need to select appropriate tasks and programming environments for each student. The key approaches to programming education in schools are characterized, including task-based, differentiated, activity-based, and competency-based approaches.

The analysis of studies on the task-based approach highlights its significance in programming education. It has been established that this approach fosters the development of algorithmic thinking, the ability to model information processes, analyze problem conditions, construct solution algorithms, and implement them using programming languages. The need for a transition from reproductive to creative programming is emphasized.

It is noted that the main challenge in programming education is the lack of a systematic approach. The importance of focusing not only on learning the syntax of programming languages, but also on applying knowledge to real-world problem-solving is underscored. An example of implementing a task-based approach for students in grades 8–11 using the Lazarus programming language is provided, incorporating flowcharts, algorithmic structures, and environments with visual execution representation.

Methods for solving programming problems are examined, including the development of an information model, algorithm construction, and implementation in a specific programming language. The stages of skill development in programming, from reproductive to creative levels, are identified. The necessity of a differentiated approach to maximize students' potential in programming education is substantiated.

Keywords: methodological approaches, algorithmization, programming, task-based approach, Lazarus, Scratch, pedagogical innovations, interactive methods, ICT competencies.

INTRODUCTION

The problem formulation. Modern education is focused on developing competencies that enable students to independently solve problems across various fields of activity. One of the key directions of STEM education is programming instruction, which fosters the development of algorithmic thinking, logic, and the ability to analyze and model processes. However, practice shows that traditional approaches to teaching programming often emphasize the study of programming language syntax and the reproductive execution of algorithms without a deep understanding of their structure and underlying principles.

One of the promising methodological strategies in programming education is the **task-based approach**, which involves acquiring knowledge and skills through solving real-world or real-world-oriented tasks. This approach not only teaches students how to write code but also enhances their ability to analyze problem conditions, design solution algorithms, and adapt programmatic solutions to different contexts. A significant challenge, however, is the lack of methodological support for the implementation of the task-based approach, the absence of a systematic classification of typical programming tasks, and the lack of well-defined criteria for assessing the effectiveness of such an approach in school education.

Moreover, modern digital tools, particularly programming environments with code execution visualization, can significantly enhance the learning process. However, their use requires thorough methodological justification. This raises the issue of developing practical methodological approaches to implementing the task-based approach in programming education, including task adaptation according to students' proficiency levels and the integration of contemporary educational technologies.

Thus, the relevance of this study is determined by the need to explore effective methodological approaches for implementing the task-based approach in school programming education. Such an approach will contribute to the development of students' algorithmic thinking, creative problem-solving skills, and the successful application of digital tools in the learning process.

Analysis of recent research and publications. In modern pedagogical science, the task-based approach to teaching programming is considered one of the key methodological tools that foster the development of algorithmic thinking and the formation of students' practical skills.

In the works of H. Kostiuk (Kostiuk, 2010) and others, the concept of thought formation is substantiated, defined as the «"task-based approach". Considering problem-solving as a goal-oriented activity, the authors place central importance on cognitive activity, during which the development of students' thinking occurs based on the specialized formation of cognitive techniques. The initial moment of the cognitive process is usually a problematic situation.

A person begins to think when they feel the need to understand something. In programming, as in mathematics, physics, chemistry, and biology, task-solving skills and abilities are developed directly through tasks problems using theoretical knowledge of the subject.

In educational and methodological literature, the concept of a task is considered from the perspective of didactics as a form of implementing educational material and a means of learning, while from the perspective of psychology, it is viewed as a goal and a stimulus for thinking. There are various definitions of the term "task" In most definitions, a task is identified as a structure consisting of several components. According to I.. Lerner, "the characteristics of any task consist of the presence of a goal dictated by a requirement or question to the problem; the necessity to consider conditions and factors that serve as prerequisites for applying a solution method and ensuring the correctness of the solution itself; and the presence or necessity of identifying and constructing a solution method" (Matsko, 2016).

H.O. Bal [3] defines a task as "a system whose essential components include: (a) the subject of the task in its initial state and (b) a model of the required state of the subject".

The well-known mathematician G. Pylya distinguishes "four stages in the problem-solving process": understanding the task statement, devising a solution plan, executing the plan, and analyzing the results» (Pylya, 1991).

N. Wirth states that "in the process of solving any problem, whether using a computer or not, it is necessary to choose a certain abstraction of reality, that is, to define a set of data that describes the real situation. This choice depends on the problem that needs to be solved; then, a way of representing this information must be chosen. In most cases, these two stages are interdependent" (Wirth Niklaus, 1976).

An analysis of psychological and pedagogical literature shows that activity theory considers the learning process as the formation of students' cognitive activity.

The theoretical foundations of this approach have been explored by Ukrainian researchers, including I. S. Mintiy (Mintiy, 2012), N. V. Morze (Morze, 2003), who emphasize the importance of using problem-solving tasks in education to enhance logical and critical thinking. Their studies highlight the necessity of adapting tasks to students' proficiency levels and integrating modern information technologies into the learning process.

Among researchers, S. Papert (Papert, 1980) made a significant contribution to programming education methodology by developing the constructivist approach using the Logo programming environment. Additionally, B. Meyer and M. Guzdial (Guzdial, 2016) investigated the integration of the task-based approach in teaching programming languages such as Python and Java. Studies by Aho A. (Aho, 2012), Arnon I. (Arnon &, 2013), Bell T. (Bell & Vahrenhold , 2018) have demonstrated that the use of contextualized tasks in programming education significantly increases students' motivation and fosters a deeper understanding of programming principles.

A review of the scientific literature confirms that the task-based approach is an effective means of developing algorithmic thinking. Moreover, its combination with modern digital technologies, such as visual programming environments (Scratch, Lazarus), greatly facilitates the learning process. In particular, educators Brusilovsky P. and Weintrop D. emphasize the importance of using interactive teaching methods, including game-based and problem-oriented tasks, which contribute to a deeper mastery of programming among secondary school students (Brusilovsky & Calabrese, 1997).

RESEARCH METHODS

The aim of this study is to identify innovative approaches to implementing methodological strategies for teaching programming to secondary school students.

The research methods include analysis and synthesis, the systems approach, comparative analysis, modeling, generalization, and empirical methods.

RESULTS OF THE RESEARCH

The study of algorithmization and programming in secondary schools follows two main directions:

1. Developing students' algorithmic thinking.
2. Mastering software development technologies.

Algorithmization should be considered a fundamental stage that precedes the study of object-oriented programming (OOP), which has become particularly relevant in the current stage of information technology development (Informatics curriculum for students in grades 5-9,2017).

The structure of the school curriculum in algorithmization and programming provides for the gradual mastery of the following topics: linear algorithms, branching algorithms, loop algorithms, and auxiliary algorithms.

When studying a specific type of algorithmic structure, it is advisable to construct a flowchart, as it visually represents the logic of the algorithm, enhances understanding of its operation, and helps form a correct perception of its functioning. Additionally, an effective means of mastering algorithmic concepts is the use of educational environments with built-in executors, such as Scratch, which allow students to model the execution process of algorithms and demonstrate different ways of constructing them.

One of the key challenges in teaching programming in schools is the lack of a systematic approach. In practice, the learning process often focuses not on developing problem-solving skills through programming or creating software products, but merely on memorizing the syntax of a specific programming language. As a result, lessons are predominantly reduced to examining isolated language constructs and performing standard exercises, which do not contribute to the formation of skills in applying programming to solve practical tasks and real-world problems. Consequently, students with a natural aptitude for programming tend to develop their skills independently, while for most students, the subject remains abstract, complex, and seemingly irrelevant to real life.

In modern education, several methodological approaches to teaching programming in secondary schools have been identified, including the task-based, differentiated, activity-based, and competency-based approaches.

It should be noted that in programming, as in mathematics, the method of problem-solving refers to a set of mental techniques, logical actions, and operations used to solve a broad class of problems. In contrast, the problem-solving approach pertains to a set of mental techniques, logical actions, and operations applied to solving a specific problem [6]. Similarly, in both programming and mathematics, nearly every problem can be solved in multiple ways (i.e., through different algorithms). A problem is considered successfully solved when the most rational approach (an efficient algorithm) is identified. A key characteristic of problem-solving in programming is that: "When developing a program, it is crucial to focus primarily on the characteristics of the situation that are directly relevant to the given problem, while disregarding factors that are considered insignificant in this particular case." (Morze, 2003).

Halperin P. distinguishes two main parts of the problem-solving process: the analytical part, which is related to studying the problem and making guesses about the ways to solve it, and is a necessary prerequisite for creative thinking, and the constructive part, which is related to constructing what "needs to be found or invented in order to obtain the solution to the problem" (Rudenko, 2017), and represents the creative component of thinking.

Solving a programming problem generally involves implementing an algorithm using a specific programming language, which requires the programmer to have both proficiency in the relevant language constructs and the ability to work within a particular programming environment.

The task-based approach to learning programming emphasizes that during the problem-solving process, the primary focus should be on constructing an information model of the problem, searching for and developing its solution algorithm, while the use and study of programming language constructs remain secondary. In this approach, a programming language serves merely as a tool for solving the problem - one that can be replaced if necessary. By shifting the emphasis away from learning syntax, this method allows students to concentrate on solving problems and helps eliminate barriers to learning new programming languages.

In the process of learning programming, four stages of problem-solving skills and abilities can be distinguished among students:

1. Reproductive Programming. – At this stage, a student is capable of solving problems only when they find a previously solved analogous problem to follow.

2. Comprehended Programming. – At this stage, a student can independently determine the data type, construct a flowchart for the problem-solving algorithm, and apply appropriate algorithmic constructs. They can write a program code, but they may not always be able to identify the most efficient algorithm.

3. Justified Programming. – At this stage, a student not only solves the problem but also provides reasoning regarding the correctness and efficiency of the designed algorithm. They analyze alternative algorithms and select the optimal one.

4. Creative Programming – At this stage, a student can independently determine methods and construct an algorithm to solve a given problem. If necessary, they acquire basic programming language syntax skills, as well as familiarity with the appropriate programming environment required for problem-solving.

Additionally, they consult various programming resources to enhance their knowledge and acquire new competencies that may be useful for solving the problem.

Practical experience suggests that at the initial stage of learning to solve simple programming problems, the teacher's primary task is to support any algorithmic approach proposed by the student, as long as it produces a correct result. At the next stage, all proposed algorithms are subjected to a collective analysis by students together with the instructor. The evaluation considers factors such as the efficiency of programming constructs used, the number of executed operations, and the amount of intermediate variables utilized. Through comparative analysis, the optimal algorithm for solving the problem is determined (Mintiy, 2012).

Let us consider an example of implementing the task-based approach for students in grades 8–11 using the Lazarus programming language.

Example Task. A queue of n customers has formed at the entrance of a store. The service time for each customer is t minutes. Determine how long it will take to serve n visitors.

Solution Steps:

1. Open the Lazarus programming environment.

2. Using the Component Palette, place the following objects on the form: three labels (TLabel), One button (TButton), Two input fields (TEdit).

3. Using the Object Inspector, set the properties for the components as follows:

- Button1 Caption: "Calculate"
- Label1 Caption: "Number of Customers"
- Label2 Caption: "Service Time per Customer"

The form should be designed as follows (Fig. 1):

4. In the Object Inspector, create an OnClick event handler for Button1 (or double-click the button).

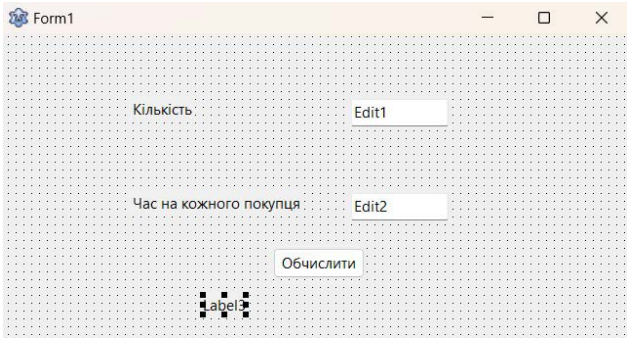

Fig 1. Placement of components on the form

5. Enter the following code in the text editor:

```
procedure TForm1.Button1Click(Sender: TObject);
var n, t, time: integer;
begin
n:=StrToInt(Edit1.Text); // Assigns the integer value entered in Edit1 as the number of store visitors
t:=StrToInt(Edit2.Text); // Assigns the integer value entered in Edit2 as the service time per customer
time:=n*t; // Calculates the total service time for all n customers and assigns it to the variable time
Label3.Caption:='Total service time for all customers = ' + IntToStr(time) + ' min';
end;
```

6. The expected output of this algorithm should result in a form similar to Figure (Fig.2):

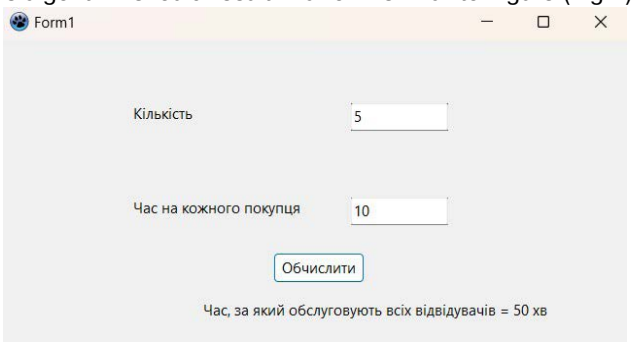

Fig 2. Program Execution Result

7. To ensure that the input fields and the output label are empty when the program starts, set the Text property of the input fields and the Caption property of the output label to an empty string ('') in the FormCreate event:

```
procedure TForm1.FormCreate(Sender: TObject);
begin
Edit1.Text:=' ';
Edit2.Text:=' ';
Label3.Caption:=' ';
end;
```

Alternatively, these property values can be set directly in the Object Inspector.

8. Finally, save the project within your folder structure, then close all windows.

To fully unleash students' potential in programming, a differentiated approach is recommended. This is because: "... students differ in their abilities, memory types, perception styles, dominant thinking patterns, and other cognitive characteristics." Different conditions, varying life experiences, differing levels of academic success in previous stages of education, and the nature of past and present teaching approaches contribute to differences among students. That is why, if we aim to maximize the development of all students in the learning process, we must apply a differentiated approach when selecting teaching forms, methods, and tools (Mintiy, 2012).

One of the means of implementing a differentiated approach in programming education is the use of a system of educational tasks with varying levels of complexity.

When selecting tasks, a focus can be placed on the algorithmic component. Problem-solving algorithms can be implemented using two programming languages: Scratch (for grades 5-7) and Lazarus (for grades 8-11).

The complexity of thematic tasks should gradually increase at each level, covering the following topics: Linear algorithms, Construction of graphic primitives, Branching (conditional algorithms), Iteration (loop algorithms).

Solving subsequent tasks requires knowledge and skills acquired from previous tasks. Tasks can be divided into three levels of complexity: Basic level – satisfactory, Intermediate level – sufficient, Advanced level – high.

Basic-level tasks are the simplest. Solving these tasks ensures that students acquire programming knowledge at a satisfactory level.

Intermediate-level tasks are more labor-intensive and complex. They require analytical thinking and a deeper understanding of the material, ensuring sufficient mastery of programming concepts.

Advanced-level tasks are the most challenging. These tasks are often comparable in difficulty to programming competition problems. They require a creative approach and genuine interest in programming, fostering a high level of proficiency.

Differentiated instruction, implemented through a structured system of tasks of varying complexity that considers students' intellectual abilities, enhances the efficiency and quality of programming education.

Experience demonstrates that the application of level-based differentiation fosters students' creative approach to learning, encourages them to strive toward their goals, enhances cognitive activity, and maintains interest in education among all learners. Additionally, it helps develop self-confidence in their abilities, knowledge, and skills - an essential factor for future professionals who will operate in a highly competitive intellectual labor market.

Experimental research confirms the feasibility and effectiveness of this approach in teaching the fundamentals of programming. Its implementation enables the individualization of the learning process, allowing for a more objective assessment of each student's knowledge level.

Multilevel learning outcome planning ensures a high degree of student autonomy in the educational process. Moreover, an individual work pace contributes to deeper comprehension and retention of knowledge, while also developing self-education and self-monitoring skills.

Below are several examples of differentiated programming tasks for students in grades 8-11, using the Lazarus programming language.

Example of a Level 1 Task on "Constructing Basic Graphic Primitives". Task: Write a program that displays an image on a form. The project should include two buttons to toggle the visibility of the image on the form (Visible = True or False).

Steps to Complete the Task:

1. Launch the Lazarus programming environment.
2. Using the Component Palette, place the following objects on the form: Two buttons (TButton), An image (TImage).
3. Using the Object Inspector, set the properties of the buttons as follows:

- Button1 Caption: "Show"
- Button2 Caption: "Hide"
- Set the properties of Image1: Height: 200 px, Width: 300 px, Visible: False

4. In the Object Inspector, create event handlers for the OnClick event of Button1 and Button2 (or double-click the button).
5. Enter the following code in the test editor window:

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  Image1.Visible:=True;
end;
procedure TForm1.Button2Click(Sender: TObject);
begin
  Image1.Visible:=False;
end;
```

6. The result of this algorithm should be a form of this type (Fig. 3).

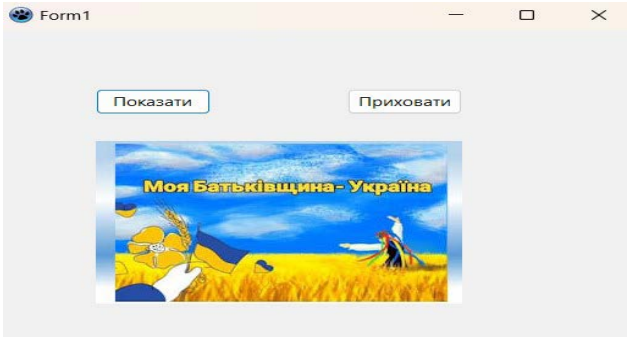

Fig. 3. Performance result

7. Save the project in its own folder structure. Close all windows.

An example of a task of the second level of complexity on the topic "Construction of basic graphic primitives". Create a "Car fleet" project, in which, when hovering the mouse over the name of the car, the corresponding images of the cars (Ferrari, Audi, BMW, Lamborghini) should be displayed.

Work progress:

1. Download the Lazarus programming environment.

2. Using the Component Panel, place the following objects on the form: four labels (TLabel) and four images (TImage).

3. Using the Object Inspector, set the properties for the labels and images as follows:

- For the image Image1 Proportional: True (similarly for the other images); Picture: load the corresponding picture with the car (similarly for the others);
- For the label Label1 Color: clRed (similarly for the others);
- For the label Label1 Caption: Ferrari (similarly for the others);

4. In the Object Inspector window, create OnMouseMove event handlers for each of the labels (there should be four instances in total) and an OnCreate event handler for the form.

5. In the text editor window, enter the code for the form that will hide all images when it is created.

```
procedure TForm1.FormCreate(Sender: TObject);
begin
  Image1.Visible:=False;
  Image2.Visible:=False;
  Image3.Visible:=False;
  Image4.Visible:=False;
end;
```

6. Next, you need to enter the code for the labels in the created event handler OnMouseMove.

```
procedure TForm1.Label1MouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
 begin
   Image1.Visible:=True;
   Image2.Visible:=False;
   Image3.Visible:=False;
   Image4.Visible:=False;
 end;
```

7. Duplicate this code for other labels, changing only the visibility of the desired images.

8. The result of executing this algorithm should be a form like this (Fig. 4).

Fig. 4. Execution result

9. Save the project in its own folder structure. Close all windows.

An example of a third-level task on the topic "Construction of basic graphic primitives". Write a program that illustrates the solution to the following task. Display an arbitrary geometric shape (for example, an ellipse) on the screen, when you click on it with the mouse, it is painted with a color that the user selects manually (from a group of switches) among such possible options as: red, blue, green.

Work progress:

1. Download the Lazarus programming environment.
2. Using the Component Panel, place the following objects on the form: radio button (TRadioGroup), shape (TShape).
3. Using the Object Inspector, set the properties for the radio buttons and shape as follows:

For Shape1 Shape: "stCircle";

For RadioGroup1 Caption: "Select a color:".

4. For the RadioGroup1 Items selection button, fill in the lines with the values: Red, Green, Blue;
5. In the Object Inspector window, create OnMouseDown event handlers for the ellipse shape Shape1.
6. In the text editor window, enter the following code for the created event handler:

```
procedure TForm1.Shape1MouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y: Integer);
begin
  if RadioGroup1.ItemIndex=0 Then Shape1.Brush.Color:=clRed;
  if RadioGroup1.ItemIndex=1 Then Shape1.Brush.Color:=clGreen;
  if RadioGroup1.ItemIndex=2 Then Shape1.Brush.Color:=clBlue;
end;
```

7. The result of executing this algorithm should be a form like this (Fig. 5).
8. Save the project in its own folder structure. Close all windows

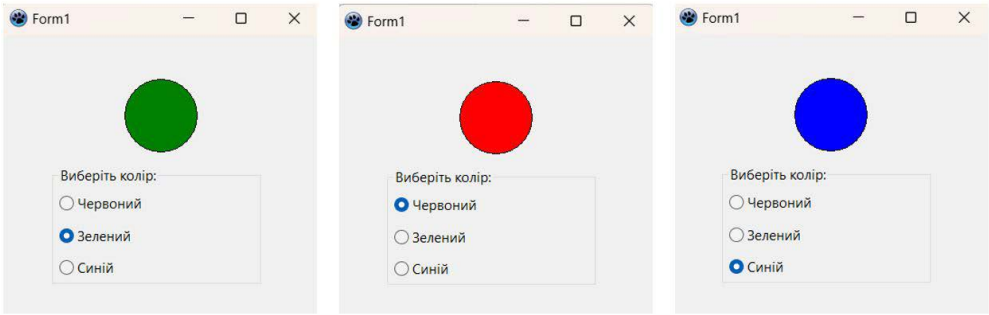

Fig. 5. Execution result

One of the key challenges in programming education is the diverse initial level of students' preparedness in informatics, particularly in the fields of algorithmization and programming. To ensure sustained interest in learning programming, it is essential to implement an individualized approach to designing educational and methodological tasks. The individualization of the educational process is achieved by enabling students to become active participants in learning, allowing them to independently determine their work pace, choose the complexity of tasks, acquire new material, and apply the acquired knowledge in practice. An important aspect of this process is objective assessment, which accurately reflects the actual level of material comprehension. Improving the effectiveness of the learning process is possible only if the individual characteristics of each student are taken into account and utilized as a tool for optimizing education.

According to the disciplinary approach, knowledge acquisition occurs through students' active cognitive engagement, in which the learning process simultaneously facilitates the development of both cognitive and practical competencies. In this context, the fundamental unit of activity is a subject-specific action that defines the structure of the educational process. The central element of student activity in programming education is the problem-solving process, which fosters the ability to develop efficient algorithms for solving various types of problems and implement them using programming techniques.

The application of an instructional approach in programming education requires the systematic use of methodological tools and the structured definition of their implementation components, contributing to a deeper understanding of the learning material and the development of students' algorithmic thinking (Semerikov, 2009). In the modern education system, it is crucial to integrate pedagogical innovations and effectively apply interactive teaching methods to enhance the quality and outcomes of the learning process.

Interactive teaching methods are instructional approaches that enhance student engagement and are based on their interaction with one another. Interactive learning is a process built on the interaction between students and the learning environment, grounded in the psychology of human relationships and interactions. It is a cognitive process in which knowledge is acquired through collaborative activities such as dialogue and polylogue.

Thus, interactive teaching methods aim at "personal development." At the same time, they not only foster active perception and personal significance but also contribute to their development. A key distinction between interactive and traditional teaching methods is the integration of real-life experiences, as well as the development of personal and professional skills through information analysis and systematization.

Through interactive exercises, students acquire the following knowledge, skills, and competencies: the development of critical thinking; the ability to analyze and evaluate their own ideas and actions; independent construction of new knowledge; defending their opinions in frequent discussions; decision-making skills; and the ability to effectively select relevant information.

Therefore, in the process of interactive learning, students must be prepared for collaborative work and active engagement in perception and communication.

The article presents an example of an interactive exercise on the topic "Quantities (Variables and Constants), Their Properties, and Basic Types of Quantities," designed for 8th-grade students. The task involves solving an exercise created in the LearningApps.org environment in the format of a "Find the Words" game (Fig. 6), which helps reinforce theoretical knowledge and develop classification skills for different types of quantities.

In the figure, students can see a 20Ч20 table filled with letters. To the right of the table, there is a list of seven words that need to be found: constant, character, variable, logical, integer, real, quantity. The words were carefully selected according to the lesson topic and relate to fundamental concepts in the study of theoretical material. Once a word is found, it lights up in green both within the table and in the word list on the right. The exercise is considered complete when the student successfully identifies all the words (Rudenko, 2017).

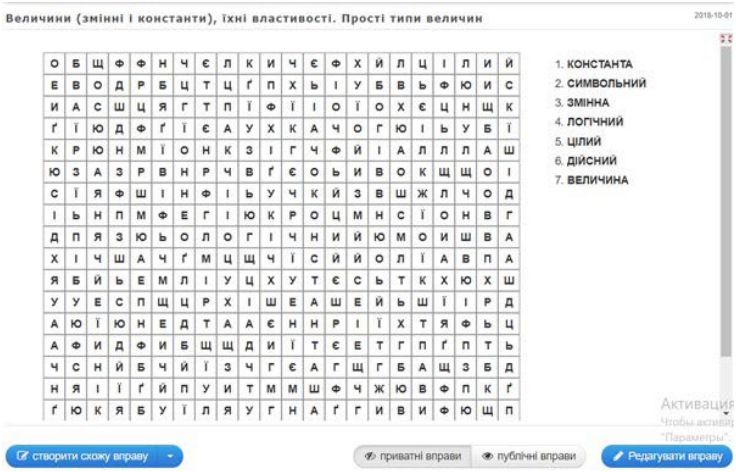

Fig. 6. Exercise "Find the words"

In the figure, students can see a 20Ч20 table filled with letters. To the right of the table, there is a list of seven words that need to be found: constant, character, variable, logical, integer, real, quantity. The words were carefully selected according to the lesson topic and relate to fundamental concepts in the study of theoretical material. Once a word is found, it lights up in green both within the table and in the word list on the right. The exercise is considered complete when the student successfully identifies all the words (Rudenko, 2017).

The implementation of a competency-based approach in the educational process enables the realization of a key concept in modern education - the interconnection between planned learning outcomes and their practical achievement. Mastering the skills required for solving competency-based tasks involves the ability to apply acquired knowledge and skills in real-world situations.

Competency formation is based on several essential factors:

- Knowledge mobility – the continuous updating of knowledge to effectively address tasks in current conditions;
- Methodological flexibility – the ability to choose an appropriate approach depending on the specific situation;
- Critical thinking – the capacity for creative and unconventional analysis.

By integrating these elements, the competency-based approach enhances students' ability to apply theoretical knowledge in practice, fostering a deeper understanding and effective problem-solving skills (Maksymenko, 2018).

The implementation of a competency-based approach in programming education is feasible under the condition that educators design specialized tasks, including:

1. Problems involving large volumes of information presented in the form of tables, graphs, charts, and diagrams;
2. Optimization problems requiring the selection of the most efficient solutions;
3. Problems with redundant or insignificant data that require analytical filtering.

The competency-based approach emphasizes the practical application of knowledge and skills in extracurricular and real-life situations (Tabl.1). The most effective methods for developing key competencies include utilizing prior experience, solving problem-based tasks, analyzing complex situations, applying discussion-based methods, engaging in game-based learning (role-playing and business simulations), and participating in project-based activities that encompass research, creativity, and practice-oriented projects.

In the field of informatics and programming, the following key types of ICT competencies can be distinguished:

- Technological competency – the ability to effectively use modern hardware and software tools to solve applied problems.
- Research competency – the ability to analyze, search for, process, and interpret information from various sources to solve ICT-related problems.
- Modeling competency – the ability to develop and utilize mathematical, informational, and logical models for analyzing and predicting processes and phenomena.
- Methodological competency – proficiency in principles and methods of organizing educational and professional activities in the field of information technology.
- Algorithmic competency – the ability to design, analyze, and optimize algorithms for automating computational processes and solving problems through programming.

By integrating these competencies into the learning process, students gain essential skills for solving real-world challenges and effectively applying programming concepts in diverse contexts [17].

Table 1

Competencies

Representative comparative and personalised works on pedagogical personalities of foreign countries and Ukraine [author's elaboration]

№	Core competencies	General subject competencies
1	**Communication competence**	**Oral dialogue**
	Dialogue "human" - "technical system"	Understanding the principles of interface design, working with dialog boxes
	Dialogue in written form	Using email for correspondence
	Work in a group	Working on a joint software project, interacting on the Internet
	Tolerance	Existence in an Internet community
2	**Information competence**	**Information search**
	Systematization, analysis and selection of information	Database design, use of various types of sorting and filtering
	Information storage	
	Information transformation	Conversion of information: from graphic to text, from analog to digital, etc.
3	**Value-meaning competence**	**Formulation of one's own learning goal**
	Decision-making, taking responsibility	Leadership in a group project, decision-making in non-standard situations
4	**Social and labor competence**	**Awareness of the existence of certain requirements for the product of one's activity**
	Analysis of the advantages and disadvantages of analogs of your own product	Design of various types, training in office technologies
5	**General cultural competence**	**Website and application design**
	Understanding the place of this science in the system of other sciences	Trends in the development of programming languages
6	**Personal development competence**	**Creating a comfortable environment and working conditions**
7	**Educational and cognitive competence**	**Ability to analyze and self-assess one's own activities**

Let us consider an example of a competency-based task for students in grades 5-7 (programming language: Scratch). Scratch is a visual programming environment that is an ideal tool for introductory programming education due to its accessibility and intuitive interface. The online version of Scratch (https://scratch.mit.edu/) allows students to create animations, games, and interactive projects without the need to install additional software. The use of this environment in the learning process helps to develop basic algorithmic skills, introduces the concepts of branching, loops, variables, and conditional operators.

The proposed game "Guess the Number" contributes to the development of logical thinking, helping students understand the principles of organizing conditional checks and loops in programming. It requires the player to interact with the program by entering numbers and receiving hints regarding the correct answer. The proposed task develops students' educational and cognitive competence.

Example of a competency task. Create a game in which you need to guess the number given by our character, which can be given in the range from 1 to 30. Use the algorithmic construction of branching and repetition to execute the algorithm.

Workflow:

1. Download the Scratch programming learning environment.
2. Create a variable called "Number". To do this, select the Variables - Create Variable command group. (Fig. 7).
3. Assign this variable a random value from 1 to 30.
4. Next, you need to “start” communication – insert the “Speak” and “Ask” blocks.
5. Use the algorithmic repetition construct (the “Repeat While” block) to repeat the following commands until the condition is met: the answer is “equal” to the Number.
6. Use the algorithmic branching construct “If – Else” from the Control command group, which will be executed until the condition is met: the answer is “greater than” the Number.

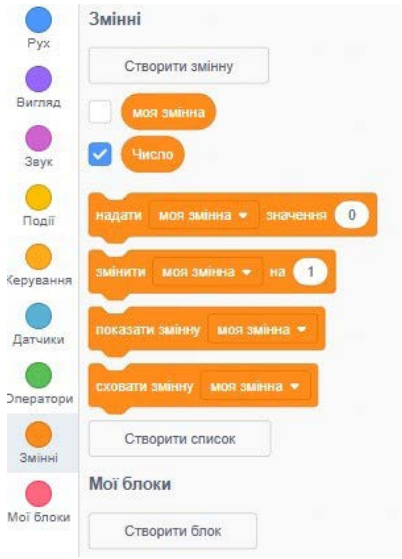

Fig. 7. Creating a variable

7. Inside the branching block, insert the “Speak” blocks, which will depend on the student’s answer to the question.
8. The result of executing this algorithm should look like this (Fig. 8).

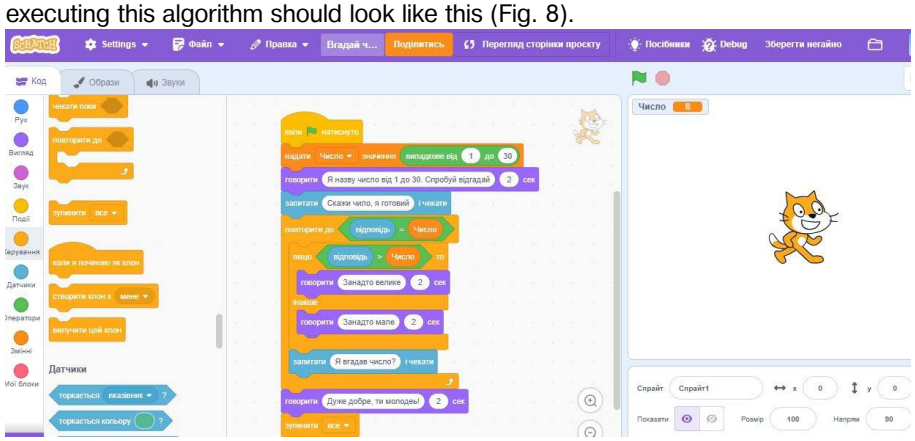

Fig. 8. Result of algorithm execution

9. Save the project in its folder structure. Close all windows.

The solution to this task not only teaches students the basic principles of programming but also enhances their creativity and problem-solving skills. The use of Scratch allows for easy modification of the game mechanics, enabling the addition of new elements such as character animation or sound effects, which fosters creative thinking [19].

Such competency-based tasks motivate students to engage in independent experimentation, help them understand the relationships between different algorithmic constructs, and contribute to the development of digital literacy, which is essential in the modern educational environment.

CONCLUSIONS AND PROSPECTS OF FURTHER RESEARCH

The article examines the task-based approach to teaching programming in schools, which is based on the integration of theoretical knowledge and practical skills through solving real-world tasks. According to the research findings, the application of this approach actively fosters the development of students’ logical thinking, creativity, and independence. The creation of tasks with varying levels of complexity and their gradual differentiation allow for consideration of individual student characteristics while ensuring a deep understanding of programming as a problem-solving process.

The methodological approaches discussed in the article include the active use of interactive platforms such as Scratch and Code.org, as well as the integration of programming with other disciplines to establish interdisciplinary connections. The practical implementation of the task-based approach in classrooms has demonstrated significant improvements in students' engagement and motivation to learn programming.

Prospects for further research include:

- Investigating the effectiveness of integrating the task-based approach with other modern teaching methods (e.g., project-based learning or gamification) to enhance students' immersion in the programming process.

- Developing new tools and platforms that meet the requirements of contemporary education and facilitate the effective implementation of the task-based approach at different stages of learning.
- Analyzing the impact of the task-based approach at the secondary education level in the context of preparing students for professional programming and their future careers in the IT industry.
- Exploring the influence of the task-based approach on the development of critical thinking and problem-solving skills, which are crucial in a rapidly evolving technological environment.

These directions will help to further assess the effectiveness of the task-based approach and its role in the advancement of modern programming education.

REFERENCES

Kostiuk H.S. – Personality, Scientist, Citizen (2010). / Edited by Acad. S.D. Maksymenko; Compiled by PhD in Psychology V.V. Andriievska. Kyiv: Nika-Center, 216 p.

URL: https://lib.iitta.gov.ua/id/eprint/6256/1/Kostyuk.indd.pdf (in Ukrainian).

Matsko V. P. (2016). Lerner Isak Yakovych. Encyclopedia of Modern Ukraine [Online] / Eds.: I. M. Dziuba, A. I. Zhukovsky, M. H. Zhelezniak [et al.] ; National Academy of Sciences of Ukraine, Shevchenko Scientific Society. Kyiv : The NASU institute of Encyclopedic Research.

URL: https://esu.com.ua/article-54403 (in Ukrainian).

Ball H.O. (2016). Psychology and Personality. №. 2 (10). Part 1, 251-253-338.

URL: http://dspace.pnpu.edu.ua/bitstream/123456789/6547/1/Ball.pdf (in Ukrainian).

Pólya G. (1991). How to Solve It. Lviv: Kvantor, 215 p.

Wirth Niklaus (1976). Algorithms+ data structures=programs. Eidgenossische Technische Hochschule Zurich, Switzerland. Prentice-hall, Inc. Englewood Cliffs, New Jersey. 366c.

URL:https://www.cl72.org/110dataAlgo/Algorithms%20%20%20Data%20Structures%20=%20Programs%20%5BWirth%201976-02%5D.pdf

Mintiy I. S. (2012). Levels of competence formation in programming. Theory and methods of teaching mathematics, physics, computer science: collection of scientific works. Issue X: in 3 volumes. Kryvyi Rih: Publishing department of NMetAU. T. 3: Theory and methods of teaching computer science. 82–86.

Morze N. V. (2003). Methods of Teaching Informatics. Part 4. Methods of Teaching the Basics of Algorithmization and Programming. Kyiv: Navchalna Knyha, 250 p.

Papert S. (1980). Mindstorms: Children, Computers, and Powerful Ideas. New York: Basic Books, 230 p.

Guzdial M. (2016). Introduction to Computing and Programming in Python: A Multimedia Approach, Fourth Edition, by Mark J. Guzdial and Barbara Ericson published by Pearson Education. 524 p.

Aho, A. V. (2012). Computation and computational thinking. The Computer Journal, 55(7), 832-835.

DOI: https://doi.org/10.1093/comjnl/bxs074

Arnon I., Cottrill J., Dubinsky E., Oktac A., Fuentes S. R., Trigueros M., Weller K. (2013). APOS theory: A framework for research and curriculum development in mathematics education. New York, NY: Springer.

DOI: https://doi.org/10.1007/978-1-4614-7966-6

Bell T. & Vahrenhold J. (2018). CS unplugged—how is it used, and does ıt work? In: Böckenhauer HJ., Komm D., Unger W. (Eds), Adventures between lower bounds and higher altitudes, 497-52. Springer, Cham.

Brusilovsky P., Calabrese E., Hvorecky J., Kouchnirenko A., & Miller, P. (1997). Mini-languages: a way to learn programming principles. Education and information technologies, 2(1), 65-83. DOI: https://doi.org/10.1023/A:1018636507883

Weintrop D., & Wilensky U. (2017). Comparing block-based and text-based programming in high school computer science classrooms. ACM Transactions on Computing Education, 18(1), 1–25.

Informatics curriculum for students in grades 5-9 who studied informatics in grades 2-4.

URL: http://mon.gov.ua/content/Новини/2017/06/12/1/8-informatika.docx.

Semerikov S. O. (2009). Fundamentalization of Teaching Informational Disciplines in Higher Education: Monograph; Scientific Editor – Academician of the Academy of Pedagogical Sciences of Ukraine, Doctor of Pedagogical Sciences, Prof. M. I. Zhaldak. Kyiv: NPU named after M. P. Drahomanov. 340 p.

Rudenko V. D. (2017). Algorithmization and Programming: Textbook; General Editor – Academician of the National Academy of Pedagogical Sciences of Ukraine, Doctor of Technical Sciences, Prof. V. Yu. Bykov. Kharkiv: Ranok Publishing House. 128 p.

Maksymenko S. D. (2018). P. Y. Halperin's theory of a stage-by-stage formation of mental actions Problems of Modern Psychology. Issue 39. 7-18.

https:// doi.org/10.32626/2227-6246.2018-39.7-18

Karavanova T. P. (2012). Informatics: Fundamentals of Algorithmization and Programming: 777 Problems with Recommendations and Applications: Textbook for Grades 8-9 with Advanced Study of Informatics / General Ed. M. Z. Zghublovskyi. Kyiv: Heneza Publishing House. 286 p.

Hromko H. Yu. (2011). Programming in Scratch: Games, Animation, Dialogues: Guidebook; Scientific Editor – PhD in Physics and Mathematics Volodymyr Lapinskyi. Kyiv: Shkilnyi Svit. 112 p.

Received *19.10.2024*
Accepted *02.11.2024*