Federal State Autonomous Educational Institution of Higher Education
Peoples' Friendship University of Russia named after Patrice Lumumba

Academy of Engineering
(name of the main educational unit (MEU) that developed the educational program of higher education)

WORKING PROGRAM OF THE DISCIPLINE

PROGRAMMING TECHNOLOGY
(name of discipline/module)

Recommended for the field of study/specialty:

27.04.04 CONTROL IN TECHNICAL SYSTEMS
(code and name of the field of study/specialty)

The discipline is mastered within the framework of the implementation of the main professional educational program of higher education (EP HE):

Artificial Intelligence, Machine Learning, and Space Science
(name (profile/specialization) of the educational institution of higher education)

2026

1. THE GOAL OF MASTERING THE DISCIPLINE

The "Programming Technology" course is part of the "Artificial Intelligence, Machine Learning, and Space Sciences" master's program, major 27.04.04 "Control in Technical Systems," and is studied in the first semester of the first year. The course is offered by the Department of Mechanics and Control Processes. It consists of 26 sections and 65 topics and focuses on basic sorting and searching algorithms, graph algorithms, dynamic programming methods, modern programming paradigms, and approaches to parallel and distributed programming technologies.

The goal of mastering the discipline is for students to acquire practical skills in algorithmization and programming.

2. REQUIREMENTS FOR THE RESULTS OF MASTERING THE DISCIPLINE

Mastering the discipline "Programming Technologies" aimed at developing the following competencies (parts of competencies) in students:

Table 2.1. List of competencies developed in students while mastering the discipline (results of mastering the discipline)

Cipher	Competence	Indicators of Competency Achievement (within this discipline)
GPC-1	Able to analyze and identify the natural scientific essence of control problems in technical systems based on provisions, laws and methods in the field of natural sciences and mathematics	GPC-1.1 Knows the basic laws, provisions and methods in the field of natural sciences and mathematics; GPC-1.2 Able to identify the natural scientific essence of control problems in technical systems guided by the laws and methods of natural sciences and mathematics; GPC-1.3 Proficient in tools for analyzing control problems in technical systems.
GPC-2	Able to formulate control problems in technical systems and justify methods for solving them	GPC-2.1 Knows the basic methods of solving control problems in technical systems; GPC-2.2 Able to justify methods for solving control problems in technical systems; GPC-2.3 Proficient in methods of setting control problems in technical systems.
GPC-3	Capable of independently solving control problems in technical systems based on the latest advances in science and technology	GPC-3.1 Knows the basic approaches to solving control problems in technical systems; GPC-3.2 Able to apply basic approaches based on the latest achievements of science and technology to solving control problems in technical systems; GPC-3.3 Proficient in methods of solving control problems in technical systems based on the latest achievements of science and technology.

3. PLACE OF THE DISCIPLINE IN THE STRUCTURE OF THE EDUCATIONAL EDUCATIONAL INSTITUTION

Discipline "Programming Technologies" refers to the mandatory part of block 1 “Disciplines (modules)” of the educational program of higher education.

As part of the higher education program, students also master other disciplines and/or practices that contribute to the achievement of the planned results of mastering the discipline "Programming Technologies".

Table 3.1. List of components of the educational program of higher education that contribute to the achievement of the planned results of mastering the discipline

Cipher	Name of competence	Previous courses/modules, practical training*	Subsequent disciplines/modules, practices*
GPC-1	Able to analyze and identify the natural scientific essence of control problems in technical systems based on provisions, laws and methods in the field of natural sciences and mathematics		Undergraduate Training; Advanced Methods of Space Flight Mechanics; Advanced Methods of Earth Remote Sensing; Geoinformation Systems and Applications;
GPC-2	Able to formulate control problems in technical systems and justify methods for solving them		Undergraduate Training; Dynamics and Control of Space Systems;
GPC-3	Capable of independently solving control problems in technical systems based on the latest advances in science and technology		Dynamics and Control of Space Systems; Advanced Methods of Space Flight Mechanics; Research work / Scientific research work; Undergraduate Training;

* - filled in accordance with the competency matrix and the SUP EP HE

** - elective courses/practices

4. SCOPE OF THE DISCIPLINE AND TYPES OF EDUCATIONAL WORK

The total workload of the “Programming Technologies” discipline is 8 credit units.

Table 4.1. Types of educational work by periods of mastering the educational program of higher education for full-time education.

Type of academic work	**TOTAL,**academic hours		**Semester(s)**
			1
Contact work, academic hours	*34*		*34*
Lectures (LC)	17		17
Laboratory work (LW)	17		17
Practical/seminar classes (SC)	0		0
Independent work of students, academic hours	*218*		*218*
Control (exam/test with assessment), academic hours	*36*		*36*
Total complexity of the discipline	**academic hours**	***288***	*288*
	credit	***8***	*8*

5. CONTENT OF THE DISCIPLINE

Table 5.1. Content of the discipline (module) by types of academic work

Section number	Name of the discipline section	Topic Title		Topic Contents	Type of academic work*
Section 1	Basic elements of Python syntax	1.1	Basic syntax of the Python 3 language. Memory model and basic data types	Basic rules for writing Python code: indentation to delimit blocks, comments, variable naming. The language's memory model features dynamic typing and reference-based object storage. Basic data types: integers and floating-point numbers, strings, Booleans, lists, tuples, dictionaries, and sets. The difference between mutable and immutable data types.	LC, LW
		1.2	Cycles and lists. Functions.	Loop constructs: for for iterating over sequences and while for repeating until a condition is met. List manipulation methods: indexing, slicing, adding and removing elements, list comprehensions. Function creation using the def keyword, positional and named arguments, and returning a value using return. Variable scopes: local and global.	LC, LW
Section 2	Elements of the theory of algorithms	2.1	The concept of an algorithm. Turing machine. Computability. Complexity theory. Exponentiation: algorithm analysis (smart exponentiation).	Definition of an algorithm as a finite sequence of strictly defined instructions. The Turing machine as an abstract model of a computer with a tape, a read/write head, and a rule table. Concept of computability: a function is considered computable if there is an algorithm for finding it. Complexity theory with a distinction between time and space complexity. Algorithm for smart exponentiation using the binary factorization method.	LC, LW
		2.2	The knapsack problem. The greedy algorithm. The gradient descent method as an example of a greedy algorithm. The divide-and-conquer strategy. A recursive algorithm.	The knapsack problem as a choice of items with given weights and costs under a constraint on the total weight. A greedy algorithm that chooses a locally optimal solution at each step. The gradient descent method as an example of a greedy algorithm for function minimization. A "divide and conquer" strategy that breaks the problem into subproblems, solves each one, and merges the results. A recursive algorithm that calls itself to solve the subproblems.	LC, LW
Section 3	Programming Paradigms. Object-Oriented Programming	3.1	Basic principles of programming. Procedural programming.	Basic programming principles: abstraction, decomposition, modularity, and hierarchical organization. Procedural programming emphasizes the sequential execution of instructions and the isolation of procedures. Structured programming is an evolution of procedural programming, eliminating unconditional jumps.	LC, LW
		3.2	Object-oriented programming (OOP). Functional programming.	Object-oriented programming as a paradigm that organizes code around objects and their interactions. Functional programming	LC, LW

Section number	Name of the discipline section	Topic Title		Topic Contents	Type of academic work*
				emphasizes pure functions, immutable data, and the absence of side effects. A comparison of three paradigms: procedural, object-oriented, and functional.	
		3.3	OOP Features. Classes and Objects. Inheritance. OOP Implementation in Python	Features of object-oriented programming: encapsulation, inheritance, and polymorphism. Classes as templates for creating objects, and objects as instances of classes. Inheritance, which allows creating new classes based on existing ones. OOP implementation in Python with class syntax, self methods, the init constructor, and multiple inheritance.	LC, LW
Section 4	Sorting and searching algorithms	4.1	Selection Sort. Insertion Sort. Bubble Sort. Merge Sort. Quick Sort.	Selection sort, which finds the minimum element and swaps it with the current position. Insertion sort, which builds the sorted part by inserting the next element into the desired position. Bubble sort, which sequentially compares and swaps adjacent elements. Merge sort, a recursive algorithm that divides the array in half and merges the sorted halves. Quicksort, which selects a pivot element and splits the array into two parts relative to the pivot.	LC, LW
		4.2	Finding the median. Sequential search. Domain narrowing methods. Sorting in Python.	Finding the median as the element in the middle of a sorted set. Sequential search, iterating through all elements until the desired one is found. Domain narrowing methods: binary search in a sorted array with interval halving. Sorting in Python using the built-in sorted function and the sort method of lists with key and reverse parameters.	LC, LW
Section 5	Graph algorithms	5.1	Graphs and their analysis. Graph representation. Depth-first and breadth-first traversal of a graph.	Graphs as a set of vertices and edges, classified as directed and undirected. Graph representation methods: adjacency matrix, adjacency list, edge list. Depth-first traversal of a graph with recursive visits to all reachable vertices along each path. Breadth-first traversal of a graph with step-by-step expansion of the visited frontier from the starting vertex.	LC, LW
		5.2	Recovering the shortest path. The chess knight problem.	Recovering the shortest path by preserving the ancestors of each vertex during a graph traversal. The problem of moving a chess knight on a chessboard, finding the minimum number of moves. Representing the board as a graph, where the vertices are cells and the edges are the knight's possible moves.	LC, LW
		5.3	Dijkstra's Algorithm. Queue and Stack. Queue and Stack in Python	Dijkstra's algorithm for finding shortest paths from one vertex to all others in a weighted graph with non-negative weights. A queue as a first-in, first-out data structure. A stack as a last-in, first-out	LC, LW

Section number	Name of the discipline section	Topic Title		Topic Contents	Type of academic work*
				data structure. Implementing a queue using collections.deque and a stack using a regular list in Python.	
Section 6	Dynamic programming	6.1	Bellman's optimality principle. The concept of bottom-up and top-down solutions.	Bellman's optimality principle: an optimal solution to a problem can be constructed from optimal solutions to its subproblems. A top-down solution involves recursively decomposing the problem and storing the results for reuse. A bottom-up solution involves filling a table from smaller subproblems to a larger one.	LC, LW
		6.2	The Route Problem: Similarities and Differences Between Dynamic Programming and Divide-and-Conquer	The problem of finding the number of routes on a grid involves counting the number of ways to get from one point to another. The similarity between dynamic programming and the divide-and-conquer strategy lies in the division into subproblems. The difference is that in dynamic programming, the subproblems overlap, while in the divide-and-conquer strategy, they are independent.	LC, LW
		6.3	The ATM Problem: Dynamic Programming and Games	The ATM problem as a problem of dispensing a minimum amount of coins or bills. Solution using dynamic programming with the construction of optimal amounts from zero to a target value. Application of dynamic programming in games: analysis of winning and losing positions using recurrence relations.	LC, LW
Section 7	Parallel algorithms	7.1	Prerequisites. Classification of computing systems. CPU and GPU processors.	The prerequisites for the emergence of parallel computing: reaching the physical limits of clock frequency and increasing volumes of processed data. Flynn's classification of computing systems: SISD, SIMD, MISD, MIMD. Central processors with a small number of powerful cores and graphics processors with thousands of low-power cores.	LC, LW
		7.2	Characteristics of parallel algorithms. Types of non-sequential programming in Python.	Characteristics of parallel algorithms: speedup, efficiency, scalability, synchronization costs. Types of non-sequential programming in Python: multithreading for I/O operations, multiprocessing for computational tasks, asynchronous programming for concurrent execution.	LC, LW
		7.3	Processes and Threads in Python. Asynchronous Programs.	Processes as isolated program instances with their own address space. Threads as lightweight units of execution within a single process with shared memory. Asynchronous programs with an event-driven model based on async and await coroutines.	LC, LW
Section 8	Program optimization	8.1	Methods for optimizing and speeding up Python programs. Profiling Python programs.	Python program optimization methods: choosing efficient algorithms, using built-in functions, and avoiding slow constructs. Profiling as a process for measuring the execution time of indi-	LC, LW

Section number	Name of the discipline section	Topic Title		Topic Contents	Type of academic work*
				vidual sections of code. The standard cProfile module for function-level profiling.	
		8.2	The line_profiler module. Python compilation: ahead-of-time and just-in-time compilation. The Numba module.	The line_profiler module for line-by-line code profiling and identifying the slowest lines. Ahead-of-time compilation converts code to native code before execution. Just-in-time compilation compiles during program execution. The Numba module for JIT compilation of functions using decorators.	LC, LW
		8.3	Cython as a Python Extension: Specifics of Cython Program Development	Cython is a superset of Python that allows calling C functions and declaring static variable types. Features of Cython development include creating .pyx files, declaring types using the cdef keyword, and compiling into an extension module. Advantages of Cython include speed close to native C while maintaining the convenience of Python.	LC, LW
Section 9	C/C++. Introduction	9.1	C and C++: language features, history, and evolution. Machine-oriented programming languages and computer operating principles.	C as a low-level procedural language designed for systems programming. C++ as an object-oriented extension of C while maintaining backward compatibility. History and evolution: from early versions of C to modern C++20 with the addition of lambdas, smart pointers, and concepts.	LC, LW
		9.2	Code translation. Types of translation. Differences between interpreters and compilers.	Code translation is the transformation of source text into machine instructions. Types of translation include compilation with full translation before execution and interpretation with line-by-line execution. Differences between compilers and interpreters: compilers create an executable file, while interpreters execute code without a separate compilation step.	LC, LW
		9.3	Comparison of Python and C/C++ programs. C/C++ application and languages.	Comparison of Python and C/C++ in terms of execution speed, development ease, and memory management. Applications of C and C++ include systems programming, game engines, high-performance computing, and embedded systems.	LC, LW
Section 10	Basic elements of syntax	10.1	Block structure of C/C++ programs, syntactic rules for allocating blocks and their types.	A block-based structure of programs, with instructions grouped into blocks using curly braces. Syntactic rules for block delimitation: each block forms a new scope. Block types: function body, loop body, conditional statement body, anonymous blocks.	LC, LW
		10.2	Basic statements: branching (or conditional statement), loops (while, do while, and for), unconditional jump statement, multiple choice statement.	Branching via the if conditional statement and the if-else construct with logical expression testing. Loops: while with a pre-test, do while with a post-test, for with a counter. The goto unconditional jump operator for transferring control to a label within a function.	LC, LW

Section number	Name of the discipline section	Topic Title		Topic Contents	Type of academic work*
				The switch multiple-choice operator for selecting a branch based on an integer value.	
		10.3	Syntactic constructs for working with functions: declaration, definition, and call. The call stack. Comparison of goto and return	Function declaration: prototype with return type, name, and parameters. Function definition with body in curly braces. Function call by name with arguments passed. Call stack as a memory area for storing local variables and return addresses. Comparison of goto and return statements: return terminates a function, goto transfers control within a function.	LC, LW
Section 11	Arrays and pointers	11.1	Pointers and addresses. Working with pointers and addresses. Arrays as data structures: memory storage and accessing elements.	Pointers as variables storing the memory addresses of other variables. Working with pointers: the address-of operator and the dereference operator for accessing a value at an address. An array as a contiguous memory area for storing elements of the same type. Accessing array elements through indexing and pointer arithmetic.	LC, LW
		11.2	Creating static arrays. Address arithmetic	Creating static arrays with a fixed size known at compile time. Address arithmetic: addition and subtraction of pointers for memory movement. Relationship between an array name and a pointer to the first element. Calculating an element's offset using an index.	LC, LW
Section 12	Static and dynamic memory.	12.1	Rules for creating, initializing, and using static arrays. One-dimensional and multidimensional static arrays. Dynamic memory (C style).	Rules for creating static arrays: the size must be a constant expression, and memory is allocated on the stack. Static arrays are initialized upon declaration with a list of values. Single-dimensional and multidimensional static arrays with sizes specified in square brackets. Dynamic memory, C-style: malloc, calloc, realloc, and free functions from the stdlib library.	LC, LW
		12.2	Dynamic Memory (C++ Style). Functions for working with dynamic memory, memory allocation and deallocation operations.	Dynamic memory in C++ style: the new operator for memory allocation and the delete operator for deallocation. The new operator with initialization values in parentheses. The delete operator for single objects and delete[] for arrays.	LC, LW
		12.3	Creating one-dimensional and multi-dimensional dynamic arrays	Creating a one-dimensional dynamic array using the new operator with a specified size. Creating a multidimensional dynamic array as an array of pointers, then allocating memory for each row. Freeing the multidimensional dynamic array in reverse order: first the rows, then the array of pointers.	LC, LW
Section 13	Structured data types	13.1	Arrays, character strings, structures, unions, enumerated data types, bit fields. Syntactic features of declaration, initialization, and operation.	Arrays as sets of identical elements with indexed access. Strings as null-terminated char arrays. Structures for combining different types of data under a single name. Unions for storing different types in a single memory area. The enum type for naming integer	LC, LW

Section number	Name of the discipline section	Topic Title		Topic Contents	Type of academic work*
				constants. Bit fields for saving memory when storing flags.	
		13.2	Memory "packaging" features. Usage examples. Dynamic data structures: vector, queue (stack), list, as examples of dynamic work with structured data.	Memory structure packing features: field alignment for efficient access, possible gaps between fields. Dynamic data structures: vector with automatic expansion when full. Stack queue with FIFO service discipline. List with nodes linked by pointers.	LC, LW
Section 14	Error interception	14.1	Exception handling syntax and usage examples.	Exception handling syntax in C++ with try, catch, and throw blocks. The try block wraps code where an error might occur. The catch block intercepts and handles an exception of a specific type. The throw statement throws an exception and passes the exception value.	LC, LW
Section 15	Data input/output	15.1	The concept of a stream and buffer. The keyboard, screen, and file as sources and destinations of data. Organizing input and output data streams in C++.	A stream is an abstraction of a sequence of data transferred between a program and a device. A buffer is a temporary storage device for I/O operations to improve efficiency. The keyboard, screen, and file are typical sources and destinations of data. Organizing I/O streams in C++ using the iostream, ifstream, and ofstream classes.	LC, LW
		15.2	Writing data to and reading data from a stream. Positioning data within a stream. File modes: read/write, character/text format, and their combinations.	Writing data to a stream using the insert operation. Reading data from a stream using the extract operation. Positioning data in a stream using the seekg function for input and seekp for output. File modes: reading, writing, appending, character text format, binary format, and combinations thereof.	LC, LW
		15.3	Text and binary files and their data storage features. Direct access files	Text files store data as characters, making them human-readable. Binary files store data in the same format as in memory, making them more compact and faster to read. Direct access files allow users to navigate to any record without reading previous records.	LC, LW
Section 16	Object-oriented programming in C++	16.1	Creating classes and objects. Setting access modifiers: public, private, and protected. Friend functions and classes.	Create classes using the class keyword with a set of fields and methods. Access modifiers: public for public access, private for access only within the class, protected for access within the class and by descendants. Friend functions and classes with the friend keyword allow access to private members.	LC, LW
		16.2	The this keyword. Inheritance in C++. Virtual functions and overloading of functions and operators.	The this keyword as a pointer to the current object within class methods. Inheritance via a colon after the class name, specifying the base class. Virtual functions with the virtual keyword for dynamic polymorphism. Overloading functions with the same name but different parameters. Operator overloading via the operator keyword.	LC, LW

Section number	Name of the discipline section	Topic Title		Topic Contents	Type of academic work*
Section 17	Using libraries	17.1	STL and BOOST Overview and Examples	The STL Standard Template Library with vector, list, map, and set containers. STL algorithms: sorting, searching, copying, and transformation. Iterators for traversing container elements. The BOOST library with advanced features: smart pointers, regular expressions, multithreading, and graphs.	LC, LW
Section 18	Parallel algorithms and systems	18.1	Classification of computing systems. CPU and GPU processors. Characteristics of parallel algorithms.	Flynn's classification of computing systems into four categories. CPUs with architecture optimized for sequential computing. GPUs with massively parallel computing for processing data streams. Characteristics of parallel algorithms: Amdahl acceleration, resource efficiency, and scalability with data and processor growth.	LC, LW
		18.2	Types of non-sequential programming. Parallel computing standards: communication between supercomputer nodes, communication between CPU cores within a single node, accelerators within a single node	Types of non-sequential programming: parallel, concurrent, and distributed. The MPI standard for communication between supercomputer nodes via message passing. The OpenMP standard for communication between cores of a single processor via shared memory. Technologies for accelerators within a single node: CUDA and OpenACC.	LC, LW
Section 19	Algorithms in external memory	19.1	Organizing computations using a hierarchical memory structure. Buffering for reading and writing.	Hierarchical memory structure: registers, first- and second-level cache, RAM, and external storage. Computations are organized with minimal access to slow memory levels. Buffering is used when reading and writing data in blocks to reduce the number of I/O operations.	LC, LW
		19.2	Complex and dynamic data structures. Graph algorithms in external memory (BFS, DFS, connected component search, MST)	Complex and dynamic data structures adapted for disk storage. Algorithms on graphs in external memory: Breadth-first traversal (BFS) and Depth-first traversal (DFS) with optimized disk access. Finding connected graph components without loading the entire graph into memory. Construction of a minimum spanning tree (MST) for graphs that do not fit in RAM.	LC, LW
Section 20	OpenMP technology	20.1	Parallel computing using the OpenMP standard.	Parallel computing using the OpenMP standard for multithreaded programming in C, C++, and Fortran. Basic information about the shared-memory parallelism model. Threads as units of execution and processes as containers for threads.	LC, LW
		20.2	Basics. Threads and processes. Parallel and serial regions.	Parallel regions with the parallel directive to create a thread pool. Serial regions executed only by the main thread. The single directive to execute a block of code by a single thread. The master directive to execute a block of code by the main thread.	LC, LW
		20.3	Parallel loops and parallel regions. Automatic	Parallel loops with the for directive for distributing iterations	LC, LW

Section number	Name of the discipline section	Topic Title		Topic Contents	Type of academic work*
			loop parallelization.	among threads. The parallel for directive combines the creation of a parallel region and a parallel loop. Automatic parallelization of loops by the compiler with data dependency analysis. The schedule clause controls the iteration distribution method.	
Section 21	MPI technology	21.1	Parallel Computing Using the MPI Standard. Basic Information. Basic MPI Procedures.	Parallel computing using the MPI standard for distributed message-passing systems. Basic information about the shared-memory process communication model. Key MPI procedures: initialization (MPI_Init), completion (MPI_Finalize), determining the process rank (MPI_Comm_rank), and determining the number of processes (MPI_Comm_size).	LC, LW
		21.2	MPI data types. Message passing methods. Processes receiving and sending messages	MPI data types for describing message contents: base types MPI_INT, MPI_FLOAT, MPI_DOUBLE, and derived types. Message transmission methods: blocking operations MPI_Send and MPI_Recv with wait for completion. Non-blocking operations MPI_Isend and MPI_Irecv with readiness checks via MPI_Wait or MPI_Test. Collective operations: MPI_Bcast for broadcasting, MPI_Reduce for reduction, MPI_Barrier for synchronization.	LC, LW
Section 22	OpenACC technology	22.1	Parallel computing using the OpenACC standard	Parallel computing using the OpenACC standard for programming GPU accelerators. Compiler directives for offloading computations to the device. Support for C, C++, and Fortran.	LC, LW
		22.2	An overview of GPU performance in various applications. Comparison of computing accelerators. Key principles for achieving high performance.	A review of GPU performance in applications such as deep learning, scientific computing, and image processing. A comparison of computing accelerators from different manufacturers based on peak performance and energy efficiency. Key principles for achieving high performance include memory coalescing and avoiding thread desynchronization.	LC, LW
		22.3	Advantages of OpenACC. Execution model: gangs, workers, vectors. Directives: parallel, kernels, loop	OpenACC's advantages over low-level approaches include: less code and portability across different accelerators. An execution model with three levels of parallelism: gangs (block groups), workers (subgroups), and vectors (vector threads). The parallel directive defines a parallel region. The kernels directive automatically parallelizes a section of code. The loop directive specifies the parallelism levels (gang, worker, and vector).	LC, LW
		22.4	Data attributes. Data regions: data, enter data, exit data. Additional data management constructs: cache, update, declare	Data attributes for controlling data movement between the host and device. Data regions: the data directive for long-term data persistence on the device. The enter data and exit data directives for	LC, LW

Section number	Name of the discipline section	Topic Title		Topic Contents	Type of academic work*
				manual control of data presence. Additional constructs: cache for caching data in shared memory, update for updating data, and declare for global variables.	
		22.5	Asynchronous execution - async and wait. Atomic operations. Global variables. OpenACC in C++.	Asynchronous execution with async and wait directives for overlapping computations and data transfer. Atomic operations for data race protection during concurrent access. Global variables with declare directives for device use. OpenACC implementation in C++ with support for classes, templates, and the standard library.	LC, LW
Section 23	CUDA hardware and software architecture	23.1	GPU architecture. GPU memory hierarchy. CUDA programming model.	GPU architecture with multiple streaming multiprocessors, each containing multiple CUDA cores. GPU memory hierarchy: global memory for all threads, shared memory within a block, local memory for an individual thread, and registers. CUDA programming model with a grid of blocks, thread blocks, and individual threads. Kernel functions with the "global" qualifier are executed on the GPU.	LC, LW
		23.2	Using C++ Libraries for OpenCL and CUDA Programming	Using C++ libraries for OpenCL programming: wrapper libraries to simplify working with platforms, devices, and command queues. Using CUDA libraries: cuBLAS for linear algebra, cuFFT for fast Fourier transform, Thrust for container algorithms. Comparison of OpenCL as a cross-platform standard and CUDA as a proprietary NVIDIA technology.	LC, LW
Section 24	Introduction to Distributed Object Technologies	24.1	The concept of a distributed information processing system. Types and properties of distributed systems.	The concept of a distributed system is a collection of independent computers interacting over a network. Types of distributed systems: client-server, peer-to-peer, and heterogeneous. Properties of distributed systems: scalability, fault tolerance, transparency, and openness.	LC, LW
		24.2	Software architecture of information systems. Managing the interaction of heterogeneous applications	Software architecture for information systems, separated into presentation, business logic, and data access layers. Managing the interaction of heterogeneous applications through common protocols and data formats. Heterogeneity issues: operating systems, programming languages, data structures.	LC, LW
		24.3	Basic mechanisms of distributed object technologies.	Basic distributed object mechanisms: remote procedure calls, remote object activation, parameter marshalling and unmarshalling. Client-to-remote object association via a name registry. Distributed object lifecycle management.	LC, LW
Section	Basic models of distributed	25.1	Types of distributed applications. Cloud tech-	Types of distributed applications: web applications, enterprise	LC, LW

Section number	Name of the discipline section	Topic Title		Topic Contents	Type of academic work*
25	object technologies		nologies. Definition of cloud computing. Multilayer architecture of cloud applications. Cloud application components.	information systems, real-time systems. Cloud technologies with the provision of computing resources as a service. Cloud computing is defined by five key characteristics: self-service, broad network access, resource pooling, rapid scaling, and measurability. Multilayered cloud application architecture: infrastructure as a service, platform as a service, and software as a service.	
		25.2	Advantages and disadvantages of cloud computing. The most common cloud platforms. GRID technologies.	Advantages of cloud computing: reduced capital expenditures, elasticity, pay-as-you-go pricing. Disadvantages: network latency, security issues, provider dependency. The most common cloud platforms are Amazon Web Services, Microsoft Azure, and Google Cloud Platform. Grid technologies are used to consolidate distributed computing resources into a single system.	LC, LW
		25.3	GRID architecture. GRID standards. Parametric GRID performance models. Comparison of GRID and cloud computing.	GRID architecture with layers: resource fabric layer, connection layer, and shared services layer. GRID standards from the Open Grid Forum. Parametric GRID performance models for throughput assessment. Comparison of GRID and cloud computing: GRID is focused on batch processing, while cloud computing focuses on on-demand services.	LC, LW
Section 26	Application integration issues	26.1	Application Integration Challenges. Complex Application Integration. Message Brokers. The Publish/Subscribe Model.	Application integration challenges: platform heterogeneity, incompatible data formats, and duplicated information. Comprehensive application integration as a systematic approach to unifying enterprise information systems. Message brokers as an intermediary for asynchronous data transfer between applications. A publish-subscribe interaction model with separation of senders and receivers through message topics.	LC, LW
		26.2	Workflow management systems. Application servers.	Workflow management systems for coordinating multi-step business processes. Task routing between participants, deadline monitoring, and event logging. Application servers as a platform for running enterprise applications with support for transactions, security, clustering, and database connectivity.	LC, LW

* - to be completed only for FULL-TIME education: LC – lectures; LW – laboratory work; SC – practical/seminar classes.

6. LOGISTIC AND TECHNICAL SUPPORT OF DISCIPLINE

Table 6.1. Material and technical support for the discipline

Audience type	Equipment of the auditorium	Specialized educational/laboratory equipment, software and materials for mastering the discipline (if necessary)
Lecture	A lecture hall equipped with specialized furniture, a whiteboard (screen), and multimedia presentation equipment.	
Computer class	A computer room for conducting classes, group and individual consultations, ongoing monitoring and midterm assessment, equipped with personal computers (in the amount of ____ units), a board (screen) and technical means for multimedia presentations.	
For independent work	A classroom for independent student work (can be used for seminars and consultations), equipped with a set of specialized furniture and computers with access to the Electronic Information System.	

* - the classroom for independent work of students MUST be indicated!

7. EDUCATIONAL, METHODOLOGICAL AND INFORMATIONAL SUPPORT OF THE DISCIPLINE

Main literature:

1. Python 3. The Essentials. Prokhorenok N., Dronov V., BHV-Petersburg, 2019 – 610 p.;

2. Python. Express course. Seder N., St. Petersburg: Piter, 2019 – 480 p.;

3. Algorithms. Handbook with examples in C, C++, Java and Python. Heineman J., Pollis G., Selkov S., St. Petersburg: Alfa-book LLC, 2017 – 432 p.;

4. High-Level Language Programming. C/C++. Khabibullin I.Sh., St. Petersburg: BHVPetersburg, 2006 – 512 p.

5. C-C++. Programmer's Handbook. G. Schildt, Williams, 2003 - 429 p.;

6. C++ Programming in Visual Studio 2010 Express. Prokhorenok N.A., 2010 – 71 p.

7. C Programming Language Brian W. Kernighan, D.M. Ritchie, Williams, 2015 – 288 s.

8. C++ programming language. Stroustrup B., Martynov N.N., Moscow: Binom, 2011. - 1135 p.

9. Parallel computing. Voevodin V. V., Voevodin Vl. V., St. Petersburg: BHV-Petersburg, 2002

10. Parallel and Distributed Programming Using C++. Hughes K., Hughes T., M.: Williams Publishing House, 2004;

11. Parallel programming using OpenMP. Levin M.P. Moscow: Binom. Knowledge Laboratory, 2008

12. Parallel programming with OpenACC. Farber R., Newnes, 2016 – 316With.;

13. CUDA Technology by Example: An Introduction to GPU Programming. Sander J., Candrot E. M.: DMK Press, 2011 - 232 p.

14. Distributed Systems. Principles and Paradigms. Tanenbaum E., van Steen M. St. Petersburg: Piter, 2003. - 877 p.

15. Developing Distributed Applications on the .NET Framework. M. Sara, R. Bill, H. Shannon, B. Mark. St. Petersburg: Piter, 2008. - 608 p.

Further reading:

1. Automating Simple Tasks with Python: A Practical Guide for Beginners. Sweirart, E., Moscow: "ID Williams", 2017 – 592 p.

2. Numerical methods: Computational practical course. Vabishchevich P.N., M.: "LIBROKOM", 2010 - 320 p.;

3. Programming language C. Lectures and exercises. S. Prata, M.: Williams Publishing House, 2013 – 960 p.;

4. C++. Sacred knowledge. Dewhurst S., St. Petersburg: Symbol Plus, 2012 – 240 p.

5. Algorithms. Handbook with examples in C, C++, Java, and Python. Heineman J., Pollis G., Selkov S., St. Petersburg: Alfa-Kniga LLC, 2017 – 432 p.

6. Algorithms construction, analysis and implementation in the C programming language. Vorozhtsov A.V., Vinokurov N.A., Moscow: MIPT, 2007 – 452 p.

7. Programming and computer science. Antonyuk V.A., Ivanov A.P., Moscow: Faculty of Physics. Moscow State University named after M. V. Lomonosova, 2015 – 64 p.

8. Sequential and parallel algorithms. Miller R., Boxer L. M.: Binom. Laboratory of knowledge, 2006

9. Introduction to parallel methods of problem solving. Yakobovsky M. V. M.: Moscow University Publishing House, 2013 – 328 p.;

10. Debugging applications for Microsoft .NET and Microsoft Windows. R. John. M.: Microsoft Press. Russian Edition. 2008.

11. XML. New perspectives for the WWW. Bumfrey F, Direnzo O, Duckett J. Publisher: "DMK Press", 2006. - 688 p.

12. Chertovskoy V. D. Databases: Theory and Practice: A Textbook for Bachelors. Sovetov B. Ya., Tsekhanovsky V. V. M.: YURAYT, 2011. - 459 p.

Resources of the information and telecommunications network "Internet":

1. RUDN University Electronic Library System and third-party electronic library systems to which university students have access based on concluded agreements

- Electronic library system of RUDN - ELS RUDN http://lib.rudn.ru/MegaPro/Web
- Electronic Library System "University Library Online" http://www.biblioclub.ru
- EBS Yurayt http://www.biblio-online.ru
- Electronic Library System "Student Consultant" www.studentlibrary.ru
- Electronic Library System "Troitsky Bridge"

2. Databases and search engines

- electronic fund of legal and regulatory documentation http://docs.cntd.ru/
- Yandex search engine https://www.yandex.ru/
- Google search engine https://www.google.ru/
- SCOPUS abstract database http://www.elsevierscience.ru/products/scopus/

Educational and methodological materials for independent work of students in mastering a discipline/module:*

1. Lecture course on the subject "Programming Technologies".

* - all teaching and methodological materials for independent work of students are posted in accordance with the current procedure on the discipline page in TUIS!

DEVELOPER:

Associate Professor		Ivanyukhin Alexey Viktorovich
Position, DEPARTMENT	*Signature*	*Surname I.O.*

HEAD OF THE DEPART-MENT:

Head of Department		Razumny Yuri Nikolaevich
Position of the DEPARTMENT	*Signature*	*Surname I.O.*

HEAD OF THE EP HE:

Professor		Razumny Yuri Nikolaevich
Position, DEPARTMENT	*Signature*	*Surname I.O.*