МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное бюджетное образовательное учреждение высшего образования
«КУБАНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»
Факультет компьютерных технологий и прикладной математики

УТВЕРЖДАЮ
Проректор по учебной работе, качеству образования – первый проректор
_____ Хагуров Т.А.
подпись
«30» мая 2025

РАБОЧАЯ ПРОГРАММА ДИСЦИПЛИНЫ

Б1.О.36«Функциональное и рекурсивно-логическое программирование»

Направление подготовки 01.03.02 Прикладная математика и информатика

Направленность (профиль) Программирование и информационные технологии

Форма обучения очная

Квалификация бакалавр

Краснодар 2025

Рабочая программа дисциплины «Функциональное и рекурсивно-логическое программирование» составлена в соответствии с федеральным государственным образовательным стандартом высшего образования (ФГОС ВО) по направлению подготовки 01.03.02Прикладная математика и информатика.

Программу составил(и):
С. Г. Синица, доцент, канд. техн. наук
И.О. Фамилия, должность, ученая степень, ученое звание

подпись

Рабочая программа дисциплины «Функциональное и рекурсивно-логическое программирование» утверждена на заседании кафедры информационных технологий протокол №15 от «14» мая 2025г.
Заведующий кафедрой (разразработчика)
В. В. Подколзин

подпись

Рабочая программа обсуждена на заседании кафедры информационных технологий протокол №15 от «14» мая 2025г.
Заведующий кафедрой (выпускающей)
В. В. Подколзин

подпись

Утверждена на заседании учебно-методической комиссии факультета компьютерных технологий и прикладной математики протокол №4 от «23» мая 2025 г.
Председатель УМК факультета
А. В. Коваленко

подпись

Рецензенты:

Бегларян М. Е., Проректор по учебной работе, Краснодарский кооперативный институт (филиал) АНО ВО Центросоюза РФ «Российский университет кооперации»

Рубцов Сергей Евгеньевич, кандидат физико-математических наук, доцент кафедры математического моделирования ФГБОУ ВО «КубГУ»

1 Цели и задачи изучения дисциплины (модуля)

Целью курса является изучение основ функционального и рекурсивно-логического программирования.

1.2 Задачи дисциплины

Основными задачами курса является:
– освоение парадигмы функционального программирования;
– освоение парадигмы рекурсивно-логического программирования;
– знакомство с языками программирования Clojure, Prolog, Scala;
– получение опыта командной разработки.

1.3 Место дисциплины (модуля) в структуре образовательной программы

Дисциплина «Функциональное и рекурсивно-логическое программирование» относится к «Обязательная часть» Блока 1 «Дисциплины (модули)» учебного плана.

1.4 Перечень планируемых результатов обучения по дисциплине (модулю), соотнесенных с планируемыми результатами освоения образовательной программы

Изучение данной учебной дисциплины направлено на формирование у обучающихся следующих компетенций:

УК-2 **Способен определять круг задач в рамках поставленной цели и выбирать оптимальные способы их решения, исходя из действующих правовых норм, имеющихся ресурсов и ограничений**

ИД-3.УК-2 ***Использует принципы проектной методологии для решения профессиональных задач***

Знать *Методы и средства проектирования программного обеспечения с использованием функциональной и слогической парадигмы программирования*

Уметь *Анализировать входные данные*
Планировать работы в проектах в области ИТ
Использовать существующие типовые решения и шаблоны проектирования программного обеспечения с использованием функционального и рекурсивно логического подхода
Применять методы и средства проектирования программного обеспечения, структур данных, баз данных, программных интерфейсов

Владеть *Проектирование структур данных*
Качественный анализ рисков в проектах в области ИТ
Деятельность, направленная на решение задач аналитического характера, предполагающих выбор и многообразие актуальных способов решения задач
Оценка и согласование сроков выполнения поставленных задач

ИД-4.УК-2 ***Выбирает оптимальный способ решения задач, имеющихся ресурсов и ограничений, оценки рисков на основе проектного инструментария***

Знать *Методы и средства планирования и организации работы в команде*

Уметь *Использовать существующие типовые решения и шаблоны проектирования программного обеспечения на основе функционального и рекурсивно-логического программирования*

Применять методы и средства проектирования программного обеспечения, структур данных, баз данных, программных интерфейсов в языках функционального программирования
Анализировать входные данные
Планировать работы в проектах в области ИТ

Владеть *Проектирование структур данных для разработки программ в функциональном и рекурсивно-логическом стиле*
Оценка и согласование сроков выполнения поставленных задач
Качественный анализ рисков в проектах в области ИТ

ОПК-5 Способен разрабатывать алгоритмы и компьютерные программы, пригодные для практического применения

ИД-1.ОПК-5 ***Аргументировано применяет методы проектирования, разработки и реализации программных продуктов и программных комплексов в различных областях человеческой деятельности***

Знать *Технологии программирования в функциональном и рекурсивно-логическом стиле*
Возможности существующей программно-технической архитектуры
Принципы построения архитектуры программного обеспечения и виды архитектуры программного обеспечения
Методы и средства проектирования программного обеспечения
Методы и средства проектирования серверных приложений

Уметь *Вырабатывать варианты реализации требований*
Использовать существующие типовые решения и шаблоны проектирования программного обеспечения
Применять методы и средства проектирования программного обеспечения, структур данных, баз данных, программных интерфейсов

Владеть *Проектирование структур данных*
Деятельность, направленная на решение задач аналитического характера, предполагающих выбор и многообразие актуальных способов решения задач
Разработка, изменение и согласование архитектуры программного обеспечения с системным аналитиком и архитектором программного обеспечения
Проектирование программных интерфейсов
Разработка структуры программного кода ИС
Деятельность, направленная на решение задач аналитического характера, предполагающих выбор и многообразие актуальных способов решения задач

ИД-2.ОПК-5 ***Использует инструментальные, программные и аппаратные средства измерений для оценки качества программного обеспечения***

Знать *Методы и средства проектирования программного обеспечения*
Современный отечественный и зарубежный опыт в профессиональной деятельности
Отечественный и международный опыт в области функционального и рекурсивно-логического программирования
Методы проведения экспериментов и наблюдений, обобщения и обработки информации
Возможности современных и перспективных средств разработки программных продуктов, технических средств
Методологии разработки программного обеспечения и технологии программирования

Уметь	*Использовать существующие типовые решения и шаблоны проектирования программного обеспечения* *Применять методы и средства проектирования программного обеспечения, структур данных, баз данных, программных интерфейсов* *Разрабатывать документы* *Верифицировать структуру программного кода* *Применять методы анализа научно-технической информации* *Проводить оценку и обоснование рекомендуемых решений*
Владеть	*Разработка, изменение и согласование архитектуры программного обеспечения с системным аналитиком и архитектором программного обеспечения* *Проведение экспериментов в соответствии с установленными полномочиями* *Оценка времени и трудоемкости реализации требований к программному обеспечению*

2. Структура и содержание дисциплины

2.1 Распределение трудоёмкости дисциплины по видам работ

Общая трудоёмкость дисциплины составляет 2 зач. ед. (72 часов), их распределение по видам работ представлено в таблице

Вид учебной работы		Всего часов	Семестры (часы)					
			7					
Контактная работа, в том числе:		**34,2**	**34,2**					
Аудиторные занятия (всего):		**34**	**34**					
Занятия лекционного типа								
Лабораторные занятия		**34**	34					
Занятия семинарского типа (семинары, практические занятия)								
Иная контактная работа:		**0,2**	**0,2**					
Контроль самостоятельной работы (КСР)								
Промежуточная аттестация (ИКР)		**0,2**	0,2					
Самостоятельная работа, в том числе:		**37,8**	**37,8**					
Выполнение индивидуальных заданий (подготовка сообщений, презентаций)		**37,8**	**37,8**					
Подготовка к текущему контролю								
Контроль:								
Подготовка к зачету								
Общая трудоемкость	**час.**	**72**	**72**					
	в том числе контактная работа	**34,2**	**34,2**					
	зач. ед	**2**	**2**					

2.2 Структура дисциплины

Распределение видов учебной работы и их трудоемкости по разделам дисциплины.
Разделы (темы) дисциплины, изучаемые в 7 семестре

№	Наименование разделов (тем)	Количество часов				
		Всего	Аудиторная работа			Внеаудиторная работа
			Л	ПЗ	ЛР	СРС
1	2	3	4	5	6	7
1.	Функциональное программирование	43			24	19
2.	Рекурсивно-логическое программирование	28,8			10	18,8
ИТОГО по разделам дисциплины		**71,8**			**34**	**37,8**
Контроль самостоятельной работы (КСР)						
Промежуточная аттестация (ИКР)		0,2				
Подготовка к текущему контролю						
Общая трудоемкость по дисциплине		**72**				

Примечание: Л – лекции, ПЗ – практические занятия/семинары, ЛР – лабораторные занятия, СРС – самостоятельная работа студента

2.3 Содержание разделов (тем) дисциплины

2.3.1 Занятия лекционного типа

Нет.

Примечание: ЛР – отчет/защита лабораторной работы, КП - выполнение курсового проекта, КР - курсовой работы, РГЗ - расчетно-графического задания, Р - написание реферата, Э - эссе, К - коллоквиум, Т – тестирование, РЗ – решение задач.

2.3.2 Занятия семинарского типа

Нет.

Примечание: ЛР – отчет/защита лабораторной работы, КП - выполнение курсового проекта, КР - курсовой работы, РГЗ - расчетно-графического задания, Р - написание реферата, Э - эссе, К - коллоквиум, Т – тестирование, РЗ – решение задач.

2.3.3 Лабораторные занятия

№	Наименование раздела (темы)	Наименование лабораторных работ	Форма текущего контроля
1	2	3	4
1.	Функциональное программирование	Введение в Clojure. Запуск программ.	РЗ
2.	Функциональное программирование	Функции в Clojure.	РЗ
3.	Функциональное программирование	Структуры данных в Clojure.	РЗ
4.	Функциональное программирование	Функции map, reduce, filter и apply.	РЗ
5.	Функциональное программирование	Решение задач Clojure Koans.	РЗ
6.	Функциональное программирование	Решение задач Clojure Koans.	РЗ
7.	Функциональное программирование	Разбор работы и доработка программы Mire	РЗ
8.	Функциональное программирование	Доработка программы Mire, работа в команде над общим проектом.	РЗ

№	Наименование раздела (темы)	Наименование лабораторных работ	Форма текущего контроля
1	2	3	4
9.	Функциональное программирование	Доработка программы Mire, работа в команде над общим проектом.	РЗ
10.	Функциональное программирование	Основы Scala.	РЗ
11.	Функциональное программирование	Решение задач на Scala.	РЗ
12.	Функциональное программирование	Решение задач на Scala.	РЗ
13.	Функциональное программирование	Решение задач на Scala.	РЗ
14.	Рекурсивно-логическое программирование	Введение в Prolog, факты, правила, механизм вывода. Списки в Prolog. Динамическое управление фактами.	РЗ
15.	Рекурсивно-логическое программирование	Расширение SWI-Prolog для работы с грамматиками DCG. Предикат phrase, разбор примера.	РЗ
16.	Рекурсивно-логическое программирование	Реализация клиента TCP/IP.	РЗ
17.	Рекурсивно-логическое программирование	Реализация бота Mire.	РЗ

Примечание: ЛР – отчет/защита лабораторной работы, КП - выполнение курсового проекта, КР - курсовой работы, РГЗ - расчетно-графического задания, Р - написание реферата, Э - эссе, К - коллоквиум, Т – тестирование, РЗ – решение задач.

2.3.4 Примерная тематика курсовых работ (проектов)

Нет.

2.4 Перечень учебно-методического обеспечения для самостоятельной работы обучающихся по дисциплине (модулю)

№	Вид СРС	Перечень учебно-методического обеспечения дисциплины по выполнению самостоятельной работы
1	2	3
1	Изучение теоретического материала	Методические указания по организации самостоятельной работы студентов, утвержденные кафедрой информационных технологий, протокол №1 от 30.08.2019
2	Решение задач	Методические указания по организации самостоятельной работы студентов, утвержденные кафедрой информационных технологий, протокол №1 от 30.08.2019

Учебно-методические материалы для самостоятельной работы обучающихся из числа инвалидов и лиц с ограниченными возможностями здоровья (ОВЗ) предоставляются в формах, адаптированных к ограничениям их здоровья и восприятия информации:

Для лиц с нарушениями зрения:

– в печатной форме увеличенным шрифтом,

– в форме электронного документа,

– в форме аудиофайла,

– в печатной форме на языке Брайля.

Для лиц с нарушениями слуха:

– в печатной форме,

– в форме электронного документа.

Для лиц с нарушениями опорно-двигательного аппарата:

– в печатной форме,

– в форме электронного документа,

– в форме аудиофайла.

Данный перечень может быть конкретизирован в зависимости от контингента обучающихся.

3. Образовательные технологии

В соответствии с требованиями ФГОС в программа дисциплины предусматривает использование в учебном процессе следующих образовательные технологии: чтение лекций с использованием мультимедийных технологий; метод малых групп, разбор практических задач и кейсов.

При обучении используются следующие образовательные технологии:

– Технология коммуникативного обучения – направлена на формирование коммуникативной компетентности студентов, которая является базовой, необходимой для адаптации к современным условиям межкультурной коммуникации.

– Технология разноуровневого (дифференцированного) обучения – предполагает осуществление познавательной деятельности студентов с учётом их индивидуальных способностей, возможностей и интересов, поощряя их реализовывать свой творческий потенциал. Создание и использование диагностических тестов является неотъемлемой частью данной технологии.

– Технология модульного обучения – предусматривает деление содержания дисциплины на достаточно автономные разделы (модули), интегрированные в общий курс.

– Информационно-коммуникационные технологии (ИКТ) - расширяют рамки образовательного процесса, повышая его практическую направленность, способствуют интенсификации самостоятельной работы учащихся и повышению познавательной активности. В рамках ИКТ выделяются 2 вида технологий:

– Технология использования компьютерных программ – позволяет эффективно дополнить процесс обучения языку на всех уровнях.

– Интернет-технологии – предоставляют широкие возможности для поиска информации, разработки научных проектов, ведения научных исследований.

– Технология индивидуализации обучения – помогает реализовывать личностно-ориентированный подход, учитывая индивидуальные особенности и потребности учащихся.

– Проектная технология – ориентирована на моделирование социального взаимодействия учащихся с целью решения задачи, которая определяется в рамках профессиональной подготовки, выделяя ту или иную предметную область.

– Технология обучения в сотрудничестве – реализует идею взаимного обучения, осуществляя как индивидуальную, так и коллективную ответственность за решение учебных задач.

– Игровая технология – позволяет развивать навыки рассмотрения ряда возможных способов решения проблем, активизируя мышление студентов и раскрывая личностный потенциал каждого учащегося.

– Технология развития критического мышления – способствует формированию разносторонней личности, способной критически относиться к информации, умению отбирать информацию для решения поставленной задачи.

Комплексное использование в учебном процессе всех вышеназванных технологий стимулируют личностную, интеллектуальную активность, развивают познавательные процессы, способствуют формированию компетенций, которыми должен обладать будущий специалист.

Основные виды интерактивных образовательных технологий включают в себя:

– работа в малых группах (команде) - совместная деятельность студентов в группе под руководством лидера, направленная на решение общей задачи путём творческого сложения результатов индивидуальной работы членов команды с делением полномочий и ответственности;

– проектная технология - индивидуальная или коллективная деятельность по отбору, распределению и систематизации материала по определенной теме, в результате которой составляется проект;

– анализ конкретных ситуаций - анализ реальных проблемных ситуаций, имевших место в соответствующей области профессиональной деятельности, и поиск вариантов лучших решений;

– развитие критического мышления – образовательная деятельность, направленная на развитие у студентов разумного, рефлексивного мышления, способного выдвинуть новые идеи и увидеть новые возможности.

Подход разбора конкретных задач и ситуаций широко используется как преподавателем, так и студентами во время лекций, лабораторных занятий и анализа результатов самостоятельной работы. Это обусловлено тем, что при исследовании и решении каждой конкретной задачи имеется, как правило, несколько методов, а это требует разбора и оценки целой совокупности конкретных ситуаций.

Семестр	Вид занятия	Используемые интерактивные образовательные технологии	количество интерактивных часов
	ЛР	Практические занятия в режимах взаимодействия «преподаватель – студент» и «студент – студент»	10
Итого			**10**

Примечание: Л – лекции, ПЗ – практические занятия/семинары, ЛР – лабораторные занятия, СРС – самостоятельная работа студента

Темы, задания и вопросы для самостоятельной работы призваны сформировать навыки поиска информации, умения самостоятельно расширять и углублять знания, полученные в ходе лекционных и практических занятий.

Подход разбора конкретных ситуаций широко используется как преподавателем, так и студентами при проведении анализа результатов самостоятельной работы.

Для лиц с ограниченными возможностями здоровья предусмотрена организация консультаций с использованием электронной почты.

Для лиц с нарушениями зрения:

– в печатной форме увеличенным шрифтом,

– в форме электронного документа.

Для лиц с нарушениями слуха:

– в печатной форме,

– в форме электронного документа.

Для лиц с нарушениями опорно-двигательного аппарата:

– в печатной форме,

– в форме электронного документа.

Для лиц с ограниченными возможностями здоровья предусмотрена организация консультаций с использованием электронной почты.

Данный перечень может быть конкретизирован в зависимости от контингента обучающихся.

4. Оценочные и методические материалы

4.1 Оценочные средства для текущего контроля успеваемости и промежуточной аттестации

Оценочные средства предназначены для контроля и оценки образовательных достижений обучающихся, освоивших программу учебной дисциплины «название дисциплины».

Оценочные средства включает контрольные материалы для проведения **текущего контроля** в форме разноуровневых заданий и **промежуточной аттестации** в форме проекта к зачету.

Оценочные средства для инвалидов и лиц с ограниченными возможностями здоровья выбираются с учетом их индивидуальных психофизических особенностей.

– при необходимости инвалидам и лицам с ограниченными возможностями здоровья предоставляется дополнительное время для подготовки ответа на экзамене;

– при проведении процедуры оценивания результатов обучения инвалидов и лиц с ограниченными возможностями здоровья предусматривается использование технических средств, необходимых им в связи с их индивидуальными особенностями;

– при необходимости для обучающихся с ограниченными возможностями здоровья и инвалидов процедура оценивания результатов обучения по дисциплине может проводиться в несколько этапов.

Процедура оценивания результатов обучения инвалидов и лиц с ограниченными возможностями здоровья по дисциплине (модулю) предусматривает предоставление информации в формах, адаптированных к ограничениям их здоровья и восприятия информации:

Для лиц с нарушениями зрения:

– в печатной форме увеличенным шрифтом,

– в форме электронного документа.

Для лиц с нарушениями слуха:

– в печатной форме,

– в форме электронного документа.

Для лиц с нарушениями опорно-двигательного аппарата:

– в печатной форме,

– в форме электронного документа.

Данный перечень может быть конкретизирован в зависимости от контингента обучающихся.

Структура оценочных средств для текущей и промежуточной аттестации

№ п/п	Контролируемые разделы (темы) дисциплины*	Код контролируемой компетенции (или ее части)	Наименование оценочного средства	
			Текущий контроль	Промежуточная аттестация
1	Функциональное программирование	УК-2, ОПК-5	*Лабораторная работа 1-13*	*Проект*

2	Рекурсивно-логическое программирование	УК-2, ОПК-5	*Лабораторная работа 14-17*	*Проект*

Показатели, критерии и шкала оценки сформированных компетенций

Соответствие **пороговому уровню** освоения компетенций планируемым результатам обучения и критериям их оценивания (оценка: **удовлетворительно /зачтено**):

УК-2 **Способен определять круг задач в рамках поставленной цели и выбирать оптимальные способы их решения, исходя из действующих правовых норм, имеющихся ресурсов и ограничений**

ИД-3.УК-2 ***Использует принципы проектной методологии для решения профессиональных задач***

Знать *Методы и средства проектирования программного обеспечения с использованием функциональной и слогической парадигмы программирования*

Уметь *Анализировать входные данные*
Планировать работы в проектах в области ИТ
Использовать существующие типовые решения и шаблоны проектирования программного обеспечения с использованием функционального и рекурсивно логического подхода
Применять методы и средства проектирования программного обеспечения, структур данных, баз данных, программных интерфейсов

Владеть *Проектирование структур данных*
Качественный анализ рисков в проектах в области ИТ
Деятельность, направленная на решение задач аналитического характера, предполагающих выбор и многообразие актуальных способов решения задач
Оценка и согласование сроков выполнения поставленных задач

ИД-4.УК-2 ***Выбирает оптимальный способ решения задач, имеющихся ресурсов и ограничений, оценки рисков на основе проектного инструментария***

Знать *Методы и средства планирования и организации работы в команде*

Уметь *Использовать существующие типовые решения и шаблоны проектирования программного обеспечения на основе функционального и рекурсивно-логического программирования*

Применять методы и средства проектирования программного обеспечения, структур данных, баз данных, программных интерфейсов в языках функционального программирования
Анализировать входные данные
Планировать работы в проектах в области ИТ

Владеть *Проектирование структур данных для разработки программ в функциональном и рекурсивно-логическом стиле*
Оценка и согласование сроков выполнения поставленных задач
Качественный анализ рисков в проектах в области ИТ

ОПК-5 **Способен разрабатывать алгоритмы и компьютерные программы, пригодные для практического применения**

ИД-1.ОПК-5 ***Аргументировано применяет методы проектирования, разработки и реализации программных продуктов и программных комплексов в различных областях человеческой деятельности***

Знать
Технологии программирования в функциональном и рекурсивно-логическом стиле
Возможности существующей программно-технической архитектуры
Принципы построения архитектуры программного обеспечения и виды архитектуры программного обеспечения
Методы и средства проектирования программного обеспечения
Методы и средства проектирования серверных приложений

Уметь
Вырабатывать варианты реализации требований
Использовать существующие типовые решения и шаблоны проектирования программного обеспечения
Применять методы и средства проектирования программного обеспечения, структур данных, баз данных, программных интерфейсов

Владеть
Проектирование структур данных
Деятельность, направленная на решение задач аналитического характера, предполагающих выбор и многообразие актуальных способов решения задач
Разработка, изменение и согласование архитектуры программного обеспечения с системным аналитиком и архитектором программного обеспечения
Проектирование программных интерфейсов
Разработка структуры программного кода ИС
Деятельность, направленная на решение задач аналитического характера, предполагающих выбор и многообразие актуальных способов решения задач

ИД-2.ОПК-5 ***Использует инструментальные, программные и аппаратные средства измерений для оценки качества программного обеспечения***

Знать
Методы и средства проектирования программного обеспечения
Современный отечественный и зарубежный опыт в профессиональной деятельности
Отечественный и международный опыт в области функционального и рекурсивно-логического программирования
Методы проведения экспериментов и наблюдений, обобщения и обработки информации
Возможности современных и перспективных средств разработки программных продуктов, технических средств
Методологии разработки программного обеспечения и технологии программирования

Уметь
Использовать существующие типовые решения и шаблоны проектирования программного обеспечения
Применять методы и средства проектирования программного обеспечения, структур данных, баз данных, программных интерфейсов
Разрабатывать документы
Верифицировать структуру программного кода
Применять методы анализа научно-технической информации
Проводить оценку и обоснование рекомендуемых решений

Владеть
Разработка, изменение и согласование архитектуры программного обеспечения с системным аналитиком и архитектором программного обеспечения

Проведение экспериментов в соответствии с установленными полномочиями
Оценка времени и трудоемкости реализации требований к программному обеспечению

Типовые контрольные задания или иные материалы, необходимые для оценки знаний, умений, навыков и (или) опыта деятельности, характеризующих этапы формирования компетенций в процессе освоения образовательной программы

В оценочные средства входят актуальные задачи с сайта http://clojurekoans.com/ и https://www.scala-exercises.org/.

Заполнить пропуски чтобы получилось равенство:

1. Равенство

"

We shall contemplate truth by testing reality, via equality"

(= __ true)

"To understand reality, we must compare our expectations against reality"

(= __ (+ 1 1))

"You can test equality of many things"

(= (+ 3 4) 7 (+ 2 __))

"Some things may appear different, but be the same"

(= __ (= 2 2/1))

"You cannot generally float to heavens of integers"

(= __ (= 2 2.0))

"But a looser equality is also possible"

(= __ (== 2.0 2))

"Something is not equal to nothing"

(= __ (not (= 1 nil)))

"Strings, and keywords, and symbols: oh my!"

(= __ (= "hello" :hello 'hello))

"Make a keyword with your keyboard"

(= :hello (keyword __))

"Symbolism is all around us"

(= 'hello (symbol __))

"What could be equivalent to nothing?"

(= __ nil)

"When things cannot be equal, they must be different"

(not= :fill-in-the-blank __)

2. Строки

"A string is nothing more than text surrounded by double quotes"

(= __ "hello")

"But double quotes are just magic on top of something deeper"

(= __ (str 'world))

"You can do more than create strings, you can put them together"

(= "Cool right?" (str __ __))

"You can even get certain characters"

(= \C (get "Characters" __))

"Or even count the characters"

(= __ (count "Hello World"))

"But strings and characters are not the same"

```
(= __ (= \c "c"))
```

"What if you only wanted to get part of a string?"

```
(= "World" (subs "Hello World" __ __))
```

"How about joining together elements in a list?"

```
(= __ (string/join '(1 2 3)))
```

"What if you wanted to separate them out?"

```
(= "1, 2, 3" (string/join __ '(1 2 3)))
```

"Maybe you want to separate out all your lines"

```
(= [__ __ __] (string/split-lines "1\n2\n3"))
```

"You may want to make sure your words are backwards"

```
(= __ (string/reverse "hello"))
```

"Maybe you want to find the index of the first occurrence of a substring"

```
(= 0 (string/index-of "hello world" __))
```

"Or maybe the last index of the same substring"

```
(= __ (string/last-index-of "hello world, hello" "hello"))
```

"But when something doesn't exist, nothing is found"

```
(= __ (string/index-of "hello world" "bob"))
```

"Sometimes you don't want whitespace cluttering the front and back"

```
(= __ (string/trim "  \nhello world \t \n"))
```

"You can check if something is a char"

```
(= __ (char? \c))
```

```
"But it may not be"
(= __ (char? "a"))

"But chars aren't strings"
(= __ (string? \b))

"Strings are strings"
(= true (string? __))

"Some strings may be blank"
(= __ (string/blank? ""))

"Even if at first glance they aren't"
(= __ (string/blank? " \n \t  "))

"However, most strings aren't blank"
(= __ (string/blank? "hello?\nare you out there?"))
```

3. Списки

```
"Lists can be expressed by function or a quoted form"
(= '(__ __ __ __ __) (list 1 2 3 4 5))

"They are Clojure seqs (sequences), so they allow access to the first"
(= __ (first '(1 2 3 4 5)))

"As well as the rest"
(= __ (rest '(1 2 3 4 5)))

"Count your blessings"
(= __ (count '(dracula dooku chocula)))
```

```
"Before they are gone"
(= __ (count '()))

"The rest, when nothing is left, is empty"
(= __ (rest '(100)))

"Construction by adding an element to the front is easy"
(= __ (cons :a '(:b :c :d :e)))

"Conjoining an element to a list isn't hard either"
(= __ (conj '(:a :b :c :d) :e))

"You can use a list like a stack to get the first element"
(= __ (peek '(:a :b :c :d :e)))

"Or the others"
(= __ (pop '(:a :b :c :d :e)))

"But watch out if you try to pop nothing"
(= __ (try
        (pop '())
        (catch IllegalStateException e
          "No dice!")))

"The rest of nothing isn't so strict"
(= __ (try
        (rest '())
        (catch IllegalStateException e
          "No dice!")))
```

4. Векторы

```
"You can use vectors in clojure as array-like structures"
(= __ (count [42]))

"You can create a vector from a list"
(= __ (vec '(1)))

"Or from some elements"
(= __ (vector nil nil))

"But you can populate it with any number of elements at once"
(= [1 __] (vec '(1 2)))

"Conjoining to a vector is different than to a list"
(= __ (conj [111 222] 333))

"You can get the first element of a vector like so"
(= __ (first [:peanut :butter :and :jelly]))

"And the last in a similar fashion"
(= __ (last [:peanut :butter :and :jelly]))

"Or any index if you wish"
(= __ (nth [:peanut :butter :and :jelly] 3))

"You can also slice a vector"
(= __ (subvec [:peanut :butter :and :jelly] 1 3))

"Equality with collections is in terms of values"
(= (list 1 2 3) (vector 1 2 __))
```

5. Множества

```
"You can create a set by converting another collection"
  (= #{3} (set __))

  "Counting them is like counting other collections"
  (= __ (count #{1 2 3}))

  "Remember that a set is a *mathematical* set"
  (= __ (set '(1 1 2 2 3 3 4 4 5 5)))

  "You can ask Clojure for the union of two sets"
  (= __ (set/union #{1 2 3 4} #{2 3 5}))

  "And also the intersection"
  (= __ (set/intersection #{1 2 3 4} #{2 3 5}))

  "But don't forget about the difference"
  (= __ (set/difference #{1 2 3 4 5} #{2 3 5}))
```

6. Отображени

```
"Don't get lost when creating a map"
  (= {:a 1 :b 2} (hash-map :a 1 __ __))

  "A value must be supplied for each key"
  (= {:a 1} (hash-map :a __))

  "The size is the number of entries"
  (= __ (count {:a 1 :b 2}))

  "You can look up the value for a given key"
  (= __ (get {:a 1 :b 2} :b))
```

"Maps can be used as functions to do lookups"

(= __ ({:a 1 :b 2} :a))

"And so can keywords"

(= __ (:a {:a 1 :b 2}))

"But map keys need not be keywords"

(= __ ({2010 "Vancouver" 2014 "Sochi" 2018 "PyeongChang"} 2014))

"You may not be able to find an entry for a key"

(= __ (get {:a 1 :b 2} :c))

"But you can provide your own default"

(= __ (get {:a 1 :b 2} :c :key-not-found))

"You can find out if a key is present"

(= __ (contains? {:a nil :b nil} :b))

"Or if it is missing"

(= __ (contains? {:a nil :b nil} :c))

"Maps are immutable, but you can create a new and improved version"

(= {1 "January" 2 __} (assoc {1 "January"} 2 "February"))

"You can also create a new version with an entry removed"

(= {__ __} (dissoc {1 "January" 2 "February"} 2))

"Create a new map by merging"

(= {:a 1 :b 2 __ __} (merge {:a 1 :b 2} {:c 3}))

"Specify how to handle entries with same keys when merging"

(= {:a 1 :b __ :c 3} (merge-with + {:a 1 :b 1} {:b 1 :c 3}))

```
"Often you will need to get the keys, but the order is undependable"
(= (list __ __ __)
  (sort (keys { 2014 "Sochi" 2018 "PyeongChang" 2010 "Vancouver"})))

"You can get the values in a similar way"
(= (list __ __ __)
  (sort (vals {2010 "Vancouver" 2014 "Sochi" 2018 "PyeongChang"})))

"You can even iterate over the map entries as a seq"
(= {:a __ :b __}
  (into {}
      (map
       (fn [[k v]] [k (inc v)])
       {:a 1 :b 2})))
```

7. Функции

```
"Calling a function is like giving it a hug with parentheses"
(= __ (square 9))

"Functions are usually defined before they are used"
(= __ (multiply-by-ten 2))

"But they can also be defined inline"
(= __ ((fn [n] (* 5 n)) 2))

"Or using an even shorter syntax"
(= __ (#(* 15 %) 4))

"Even anonymous functions may take multiple arguments"
(= __ (#(+ %1 %2 %3) 4 5 6))
```

```
"Arguments can also be skipped"
(= __ (#(str "AA" %2) "bb" "CC"))

"One function can beget another"
(= 9 (((fn [] ___)) 4 5))

"Functions can also take other functions as input"
(= 20 ((fn [f] (f 4 5))
       ___))

"Higher-order functions take function arguments"
(= 25 (___
       (fn [n] (* n n))))

"But they are often better written using the names of functions"
(= 25 (___ square))
```

8. Условия

```
"You will face many decisions"
(= __ (if (false? (= 4 5))
        :a
        :b))

"Some of them leave you no alternative"
(= __ (if (> 4 3)
        []))

"And in such a situation you may have nothing"
(= __ (if (nil? 0)
        [:a :b :c]))
```

```
"In others your alternative may be interesting"
(= :glory (if (not (empty? ()))
          :doom
          __))

"You may have a multitude of possible paths"
(let [x 5]
  (= :your-road (cond (= x __) :road-not-taken
                      (= x __) :another-road-not-taken
                      :else __)))

"Or your fate may be sealed"
(= 'doom (if-not (zero? __)
       'doom
       'more-doom))

"In case of emergency, go fast"
(= "pretty fast"
   (explain-exercise-velocity __))

"But admit it when you don't know what to do"
(= __
   (explain-exercise-velocity :watching-tv))
```

9. Функции высокого порядка

```
"The map function relates a sequence to another"
(= [__ __ __] (map (fn [x] (* 4 x)) [1 2 3]))

"You may create that mapping"
(= [1 4 9 16 25] (map (fn [x] __) [1 2 3 4 5]))
```

```
  "Or use the names of existing functions"
  (= __ (map nil? [:a :b nil :c :d]))

  "A filter can be strong"
  (= __ (filter (fn [x] false) '(:anything :goes :here)))

  "Or very weak"
  (= __ (filter (fn [x] true) '(:anything :goes :here)))

  "Or somewhere in between"
  (= [10 20 30] (filter (fn [x] __) [10 20 30 40 50 60 70 80]))

  "Maps and filters may be combined"
  (= [10 20 30] (map (fn [x] __) (filter (fn [x] __) [1 2 3 4 5 6 7 8])))

  "Reducing can increase the result"
  (= __ (reduce (fn [a b] (* a b)) [1 2 3 4]))

  "You can start somewhere else"
  (= 2400 (reduce (fn [a b] (* a b)) __ [1 2 3 4]))

  "Numbers are not the only things one can reduce"
  (= "longest" (reduce (fn [a b]
                         (if (< __ __) b a))
                       ["which" "word" "is" "longest"]))
```

10. Мультиметоды

```
(defn hello
  ([] "Hello World!")
  ([a] (str "Hello, you silly " a "."))
```

```
  ([a & more] (str "Hello to this group: "
                   (apply str
                          (interpose ", " (cons a more)))
                   "!")))

(defmulti diet (fn [x] (:eater x)))
(defmethod diet :herbivore [a] __)
(defmethod diet :carnivore [a] __)
(defmethod diet :default [a] __)

  "Some functions can be used in different ways - with no arguments"
  (= __ (hello))

  "With one argument"
  (= __ (hello "world"))

  "Or with many arguments"
  (= __
    (hello "Peter" "Paul" "Mary"))

  "Multimethods allow more complex dispatching"
  (= "Bambi eats veggies."
    (diet {:species "deer" :name "Bambi" :age 1 :eater :herbivore}))

  "Animals have different names"
  (= "Thumper eats veggies."
    (diet {:species "rabbit" :name "Thumper" :age 1 :eater :herbivore}))

  "Different methods are used depending on the dispatch function result"
  (= "Simba eats animals."
    (diet {:species "lion" :name "Simba" :age 1 :eater :carnivore}))
```

```
"You may use a default method when no others match"
(= "I don't know what Rich Hickey eats."
   (diet {:name "Rich Hickey"}))
```

11. Ленивые последовательности

```
"There are many ways to generate a sequence"
(= __ (range 1 5))

"The range starts at the beginning by default"
(= __ (range 5))

"Only take what you need when the sequence is large"
(= [0 1 2 3 4 5 6 7 8 9]
   (take __ (range 100)))

"Or limit results by dropping what you don't need"
(= [95 96 97 98 99]
   (drop __ (range 100)))

"Iteration provides an infinite lazy sequence"
(= __ (take 8 (iterate (fn [x] (* x 2)) 1)))

"Repetition is key"
(= [:a :a :a :a :a :a :a :a :a :a]
   (repeat 10 __))

"Iteration can be used for repetition"
(= (repeat 100 "hello")
   (take 100 (iterate ___ "hello")))
```

12. Обработка последовательностей

```
"Sequence comprehensions can bind each element in turn to a symbol"
 (= __
   (for [x (range 6)]
     x))

 "They can easily emulate mapping"
 (= '(0 1 4 9 16 25)
   (map (fn [x] (* x x))
        (range 6))
   (for [x (range 6)]
     __))

 "And also filtering"
 (= '(1 3 5 7 9)
   (filter odd? (range 10))
   (for [x __ :when (odd? x)]
     x))

 "Combinations of these transformations are trivial"
 (= '(1 9 25 49 81)
   (map (fn [x] (* x x))
        (filter odd? (range 10)))
   (for [x (range 10) :when __]
     __))

 "More complex transformations simply take multiple binding forms"
 (= [[:top :left] [:top :middle] [:top :right]
    [:middle :left] [:middle :middle] [:middle :right]
    [:bottom :left] [:bottom :middle] [:bottom :right]]
   (for [row [:top :middle :bottom]
         column [:left :middle :right]]
```

```
  __))
```

13. Создание функций

"One may know what they seek by knowing what they do not seek"

```
 (= [__ __ __] (let [not-a-symbol? (complement symbol?)]
                 (map not-a-symbol? [:a 'b "c"])))
```

"Praise and 'complement' may help you separate the wheat from the chaff"

```
 (= [:wheat "wheat" 'wheat]
    (let [not-nil? ___]
      (filter not-nil? [nil :wheat nil "wheat" nil 'wheat nil])))
```

"Partial functions allow procrastination"

```
 (= 20 (let [multiply-by-5 (partial * 5)]
         (___ __)))
```

"Don't forget: first things first"

```
 (= [__ __ __ __]
    (let [ab-adder (partial concat [:a :b])]
      (ab-adder [__ __])))
```

"Functions can join forces as one 'composed' function"

```
 (= 25 (let [inc-and-square (comp square inc)]
         (inc-and-square __)))
```

"Have a go on a double dec-er"

```
 (= __ (let [double-dec (comp dec dec)]
         (double-dec 10)))
```

"Be careful about the order in which you mix your functions"

```
 (= 99 (let [square-and-dec ___]
         (square-and-dec 10)))
```

14. Рекурсия

```
"Recursion ends with a base case"
(= true (is-even? 0))

"And starts by moving toward that base case"
(= false (is-even? 1))

"Having too many stack frames requires explicit tail calls with recur"
(= false (is-even-bigint? 100003N))

"Reversing directions is easy when you have not gone far"
(= '(1) (recursive-reverse [1]))

"Yet it becomes more difficult the more steps you take"
(= '(6 5 4 3 2) (recursive-reverse [2 3 4 5 6]))

"Simple things may appear simple"
(= 1 (factorial 1))

"They may require other simple steps"
(= 2 (factorial 2))

"Sometimes a slightly bigger step is necessary"
(= 6 (factorial 3))

"And eventually you must think harder"
(= 24 (factorial 4))

"You can even deal with very large numbers"
(< 10000000000000000000000000N (factorial 1000N))
```

```
"But what happens when the machine limits you?"
(< 10000000000000000000000000N (factorial 100003N))
```

15. Деструктурирование

```
"Destructuring is an arbiter: it breaks up arguments"
(= __ ((fn [[a b]] (str b a))
      [:foo :bar]))

"Whether in function definitions"
(= (str "An Oxford comma list of apples, "
        "oranges, "
        "and pears.")
  ((fn [[a b c]] __)
   ["apples" "oranges" "pears"]))

"Or in let expressions"
(= "Rich Hickey aka The Clojurer aka Go Time aka Lambda Guru"
  (let [[first-name last-name & aliases]
        (list "Rich" "Hickey" "The Clojurer" "Go Time" "Lambda Guru")]
    __))

"You can regain the full argument if you like arguing"
(= {:original-parts ["Stephen" "Hawking"] :named-parts {:first "Stephen" :last "Hawking"}}
  (let [[first-name last-name :as full-name] ["Stephen" "Hawking"]]
    __))

"Break up maps by keys"
(= "123 Test Lane, Testerville, TX"
  (let [{street-address :street-address, city :city, state :state} test-address]
    __))
```

```
"Or more succinctly"
(= "123 Test Lane, Testerville, TX"
   (let [{:keys [street-address __ __]} test-address]
     __))

"All together now!"
(= "Test Testerson, 123 Test Lane, Testerville, TX"
   (___ ["Test" "Testerson"] test-address))
```

16. Транзакционная память

```
"In the beginning, there was a word"
(= __ (deref the-world))

"You can get the word more succinctly, but it's the same"
(= __ @the-world)

"You can be the change you wish to see in the world."
(= __ (do
        (dosync (ref-set the-world "better"))
        @the-world))

"Alter where you need not replace"
(= __ (let [exclamator (fn [x] (str x "!"))]
        (dosync
         (alter the-world exclamator)
         (alter the-world exclamator)
         (alter the-world exclamator))
        @the-world))

"Don't forget to do your work in a transaction!"
```

```
(= 0 (do __
        @the-world))

"Functions passed to alter may depend on the data in the ref"
(= 20 (do
       (dosync (alter the-world ___))))

"Two worlds are better than one"
(= ["Real Jerry" "Bizarro Jerry"]
    (do
      (dosync
       (ref-set the-world {})
       (alter the-world assoc :jerry "Real Jerry")
       (alter bizarro-world assoc :jerry "Bizarro Jerry")
       __)))
```

17. Атомы

```
"Atoms are like refs"
(= __ @atomic-clock)

"You can change at the swap meet"
(= __ (do
       (swap! atomic-clock inc)
       @atomic-clock))

"Keep taxes out of this: swapping requires no transaction"
(= 5 (do
      __
      @atomic-clock))

"Any number of arguments might happen during a swap"
```

```
(= __ (do
      (swap! atomic-clock + 1 2 3 4 5)
      @atomic-clock))

"Atomic atoms are atomic"
(= __ (do
      (compare-and-set! atomic-clock 100 :fin)
      @atomic-clock))

"When your expectations are aligned with reality, things proceed that way"
(= :fin (do
       (compare-and-set! __ __ __)
       @atomic-clock))
```

Заполнить пропуски чтобы получилось истинное выражение:

1. Типы и выражения

```
16.toHexString shouldBe

(0 to 10).contains(10) shouldBe true
(0 until 10).contains(10) shouldBe __
"foo".drop(1) shouldBe "oo"
"bar".take(2) shouldBe __
```

2. Определения и вычисления

```
def triangleArea(base: Double, height: Double): Double =
  base * height / __

triangleArea(3, 4) shouldBe 6.0
triangleArea(5, 6) shouldBe __
```

3. Рекурсия

```
def factorial(n: Int): Int =
  if (n == __) __
  else factorial(n - 1) * n

factorial(3) shouldBe 6
factorial(4) shouldBe 24
```

4. Лексическая область видимости

```
object Foo {
  val x = 1
}
```

```
object Bar {
  val x = 2
}
object Baz {
  import Bar.x
  val y = x + Foo.x
}

Baz.y shouldBe __
```

5. Хвостовая рекурсия

```
def factorial(n: Int): Int = {
  @tailrec
  def iter(x: Int, result: Int): Int =
    if (x == __) result
    else iter(x - 1, result * x)

  iter(n, __)
}

factorial(3) shouldBe 6
factorial(4) shouldBe 24
```

6. Структурирование информации

```
sealed trait Duration
case object Whole extends Duration
case object Half extends Duration
case object Quarter extends Duration

def fractionOfWhole(duration: Duration): Double =
  duration match {
    case Whole => 1.0
    case Half => __
    case Quarter => __
  }

fractionOfWhole(Half) shouldBe 0.5
fractionOfWhole(Quarter) shouldBe 0.25
```

7. Функции высокого порядка

```
def sum(f: Int => Int, a: Int, b: Int): Int = {
  def loop(x: Int, acc: Int): Int = __
    if (x > b) acc
    else loop(x + 1, acc + f(x))
  loop(a, __)
}
sum(x => x, 1, __) shouldBe 55
```

8. Стандартная библиотека

```
def triple(x: Int): Int = 3 * x

def tripleEither(x: Either[String, Int]): Either[String, Int] =
  x.map(triple)

tripleEither(Right(1)) shouldBe __
```

```
tripleEither(Left("not a number")) shouldBe __
```

9. Синтаксические улучшения

```
type Result = Either[String, (Int, Int)]
def divide(dividend: Int, divisor: Int): Result =
  if (divisor == 0) Left("Division by zero")
  else Right((dividend / divisor, dividend % divisor))
divide(6, 4) shouldBe Right((1, 2))
divide(2, 0) shouldBe Left("Division by zero")
divide(8, 4) shouldBe __
```

10. ООП

```
abstract class Reducer(init: Int) {
  def combine(x: Int, y: Int): Int
  def reduce(xs: List[Int]): Int =
    xs match {
      case Nil => init
      case y :: ys => combine(y, reduce(ys))
    }
}

object Product extends Reducer(1) {
  def combine(x: Int, y: Int): Int = x * y
}

object Sum extends Reducer(0) {
  def combine(x: Int, y: Int): Int = x + y
}

val nums = List(1, 2, 3, 4)

Product.reduce(nums) shouldBe __
Sum.reduce(nums) shouldBe __
```

11. Императивное программирование

```
def factorial(n: Int): Int = {
  var result = __
  var i = __
  while (i <= n) {
    result = result * i
    i = i + __
  }
  result
}

factorial(2) shouldBe 2
factorial(3) shouldBe 6
factorial(4) shouldBe 24
factorial(5) shouldBe 120
```

12. Функциональное программирование

```
def fib(n: Int): Int = {
  @annotation.tailrec
  def loop(n: Int, prev: Int, cur: Int): Int =
    if (n <= __) prev
```

```
    else loop(n - 1, cur, prev + cur)
  loop(n, 0, 1)
}

fib(5) should be(5)

def isSorted[A](as: Array[A], ordering: (A, A) => Boolean): Boolean = {
  @annotation.tailrec
  def go(n: Int): Boolean =
    if (n >= as.length - 1) true
    else if (!ordering(as(n), as(n + 1))) false
    else go(n + 1)

  go(0)
}

isSorted(Array(1, 3, 5, 7), (x: Int, y: Int) => x < y) shouldBe __
isSorted(Array(7, 5, 1, 3), (x: Int, y: Int) => x > y) shouldBe __
isSorted(
  Array("Scala", "Exercises"),
  (x: String, y: String) => x.length < y.length) shouldBe __
```

Проект

Студенты работают в команде над сервером многопользовательской сетевой риалтаймовой игры на базе Mire. Каждому студенту дается задание доработать сервер для добавления нескольких команд, реализующих определенные аспекты игры.

Например:

Buy – показывает список предметов для покупки в магазине.

Buy item – покупает предмет item.

Студентам дается пример клиента на SWI Prolog с использованием DCG и предлагается доработать его для поддержки всех команд.

Экзамен происходит в игровой форме в виде защиты проекта. В ходе экзамена соревнуются разработанные студентами программы на Prolog на доработанном студентами сервере Mire.

Оценка отлично ставится если студент выполнил все задачи лабораторных работ, представил доработки в сервер и разработал бота на Prolog, успешно участвующего в соревнованиях.

Оценка хорошо ставится если студент выполнил все задачи лабораторных работ, представил доработки в сервер.

Студентам, которые не справились с заданиями по работе в команде, для получения оценки удовлетворительно даются индивидуальные задания на Clojure, Scala и SWI Prolog. Для получения оценки удовлетворительно (зачтено) необходимо решить самостоятельно две задачи. Примеры индивидуальных заданий:

Списки. Дан список целых чисел. Написать функцию (предикат), истинную тогда и только тогда, когда:

1. Содержит полиндром длины k.

2. Фибоначчи > 3.

3. Арифметическая прогрессия.

4. Геометрическая прогрессия.

5. Содержит полиндром > n.

6. Содержит Фибоначчи > n.

8. Сумма четных и нечетных чисел совпадает.

9. Четные и нечетные числа чередуются.

10. Максимум встречается не более 3 раз.

11. Максимум встречается не менее 2 раз.

12. НОД всех чисел < минимума в списке.

13. НОК рядом стоящих чисел > максимума в списке.

14. Является записью двоичного кода дерева.

15. Состоит из пар чисел, лежащих на плоскости на одной прямой.

16. Состоит из троек чисел, лежащих в 3D-пространстве на одной прямой.

17. Неубывающая последовательность, с четным минимумом и нечетным максимумом.

18. Сумма четных чисел больше суммы нечетных чисел.

19. Возрастающая последовательность с нечетным минимумом и четным максимумом.

Графы. Дан неориентированный граф в виде списка ребер. Задается с клавиатуры числом вершин и парами соединенных ребрами вершин. Написать функцию (предикат), истинную тогда и только тогда, когда граф:

1. Содержит цикл заданной длины.

2. Содержит путь заданной длины.

3. Не содержит элементарный цикл заданной длины.

4. Не содержит элем. путь заданной длины.

5. Не содержит элем. путь, длиннее чем заданное число ребер.

6. Любую пару вершин соединяет путь длинны <= 5.

7. По номеру вершины вывести номера не соединённых путем с ней.

8. По номеру вершины вывести номера вершин, которые с данной соединяет путь длинны 3.

9. Даны 2 номера вершин, определить существует ли проходящий через них элементарный цикл.

10. Даны 2 номера вершин, определить соединяет ли их путь длинны 4.

11. Пары вершин с петлями не соединят путь длинны 3.

12. Даны 2 вершины, найти кратчайший путь обходом графа в ширину.

13. Даны 2 вершины, найти кратчайший путь обходом графа в глубину.

14. Даны 2 вершины, найти кратчайший элементарный цикл, содержащий их.

15. Даны 2 вершины, определить соединяет ли их путь из вершин с меньшими номерами.

16. Дан связный граф. Определить, содержит ли граф вершину, при удалении которой он становится несвязным. Вывести вершину.

16.1. Дан связный граф. Определить, содержит ли граф ребро, при удалении которого граф не содержит циклов. Вывести ребро.

17. Вывести длину максимального пути, проходящего только через вершины максимальной степени.

18. Вывести длину максимального пути, проходящего только через вершины минимальной степени без петель.

19. Вывести длину максимального пути, проходящего только через вершины с петлями.

Перечень компетенций (части компетенции), проверяемых оценочным средством

УК-2, ОПК-5.

4.2 Методические материалы, определяющие процедуры оценивания знаний, умений, навыков и (или) опыта деятельности, характеризующих этапы формирования компетенций

Оценочные средства для инвалидов и лиц с ограниченными возможностями здоровья выбираются с учетом их индивидуальных психофизических особенностей.

– при необходимости инвалидам и лицам с ограниченными возможностями здоровья предоставляется дополнительное время для подготовки ответа на экзамене;

– при проведении процедуры оценивания результатов обучения инвалидов и лиц с ограниченными возможностями здоровья предусматривается использование технических средств, необходимых им в связи с их индивидуальными особенностями;

– при необходимости для обучающихся с ограниченными возможностями здоровья и инвалидов процедура оценивания результатов обучения по дисциплине может проводиться в несколько этапов.

Процедура оценивания результатов обучения инвалидов и лиц с ограниченными возможностями здоровья по дисциплине предусматривает предоставление информации в формах, адаптированных к ограничениям их здоровья и восприятия информации:

Для лиц с нарушениями зрения:

– в печатной форме увеличенным шрифтом,

– в форме электронного документа.

Для лиц с нарушениями слуха:

– в печатной форме,

– в форме электронного документа.

Для лиц с нарушениями опорно-двигательного аппарата:

– в печатной форме,

– в форме электронного документа.

Данный перечень может быть конкретизирован в зависимости от контингента обучающихся.

5. Перечень основной и дополнительной учебной литературы, необходимой для освоения дисциплины (модуля)

5.1 Основная литература:

1. Ефимова, Е.А. Основы программирования на языке Visual Prolog / Е.А. Ефимова. - 2-е изд., испр. - Москва : Национальный Открытый Университет «ИНТУИТ», 2016. - 266 с. https://biblioclub.ru/index.php?page=book_red&id=428996&sr=1

2. Кубенский, А. А. Функциональное программирование : учебник и практикум для академического бакалавриата / А. А. Кубенский. — М. : Издательство Юрайт, 2018. — 348 с. https://biblio-online.ru/book/658E3C89-AAD5-498B-8B34-A29E1750D810/funkcionalnoe-programmirovanie

3. Рогозин О. В. Функциональное и рекурсивно-логическое программирование: учебно-методический комплекс. Москва: Евразийский открытый институт, 2009. 139 стр. ISBN: 978-5-374-00182-2 https://biblioclub.ru/index.php?page=book_view_red&book_id=90927

5.2 Дополнительная литература:

1. The Joy of Clojure, 2-ed. M. Fogus, C. Houser. ISBN: 9781617291418. Manning Publications Co.
2. Functional Programming in Scala. Paul Chiusano, Runar Bjarnason. Manning, September 2014. ISBN 9781617290657
3. Иван Братко. Язык PROLOG (Пролог): алгоритмы искусственного интеллекта 3-е издание. 640 стр., с ил.; ISBN 5-8459-0664-4, 0-201-40375-7; 2004, 3 кв.; Вильямс.
4. Программирование на JAVA [Текст] : учебное пособие / С. Г. Синица, А. В. Уварова ; М-во образования и науки Рос. Федерации, Кубанский гос. ун-т. - Краснодар : [Кубанский государственный университет], 2016. - 117 с. : ил. - Библиогр.: с. 116. - ISBN 978-5-8209-1215-3
5. Стандарты оформления исходного кода программ и современные интегрированные среды разработки программного обеспечения: учеб.-метод.пособие. Ю.В. Кольцов, А.В.Уварова, С.Г.Синица [и др.] – Краснодар: Кубанский гос.ун-т, 2017
6. Чанышев О. Г. Программирование в логике: учебное пособие Омск: Омский государственный университет, 2004. - 32 с. ISBN: 5-7779-0510-3. Электронная библиотечная система «Университетская библиотека ONLINE». URL: http://www.biblioclub.ru/book/83722/
7. Головешкин В. А. , Ульянов М. В. Теория рекурсии для программистов. Учебное пособие М.: Физматлит, 2006. - 146 с. ISBN: 978-5-9221-0721-1. Электронная библиотечная система «Университетская библиотека ONLINE». URL: http://www.biblioclub.ru/book/76680/

5.3. Периодические издания:

1. Базы данных компании «Ист Вью» http://dlib.eastview.com
2. Электронная библиотека GREBENNIKON.RU https://grebennikon.ru/

5.4. Интернет-ресурсы, в том числе современные профессиональные базы данных и информационные справочные системы

Электронно-библиотечные системы (ЭБС):

- ЭБС «ЮРАЙТ» https://urait.ru/
- ЭБС «УНИВЕРСИТЕТСКАЯ БИБЛИОТЕКА ОНЛАЙН» http://www.biblioclub.ru/
- ЭБС «BOOK.ru» https://www.book.ru
- ЭБС «ZNANIUM.COM» www.znanium.com
- ЭБС «ЛАНЬ» https://e.lanbook.com

Профессиональные базы данных

3 Scopus http://www.scopus.com/
4 ScienceDirect https://www.sciencedirect.com/
5 Журналы издательства Wiley https://onlinelibrary.wiley.com/
6 Научная электронная библиотека (НЭБ) http://www.elibrary.ru/
7 Полнотекстовые архивы ведущих западных научных журналов на Российской платформе научных журналов НЭИКОН http://archive.neicon.ru

8 Национальная электронная библиотека (доступ к Электронной библиотеке диссертаций Российской государственной библиотеки (РГБ) https://rusneb.ru/
9 Президентская библиотека им. Б.Н. Ельцина https://www.prlib.ru/
10 База данных CSD Кембриджского центра кристаллографических данных (CCDC) https://www.ccdc.cam.ac.uk/structures/
11 Springer Journals: https://link.springer.com/
12 Springer Journals Archive: https://link.springer.com/
13 Nature Journals: https://www.nature.com/
14 Springer Nature Protocols and Methods: https://experiments.springernature.com/sources/springer-protocols
15 Springer Materials: http://materials.springer.com/
16 Nano Database: https://nano.nature.com/
17 Springer eBooks (i.e. 2020 eBook collections): https://link.springer.com/
18 "Лекториум ТВ" http://www.lektorium.tv/
19 Университетская информационная система РОССИЯ http://uisrussia.msu.ru

Информационные справочные системы

1. Консультант Плюс - справочная правовая система (доступ по локальной сети с компьютеров библиотеки)

Ресурсы свободного доступа

3 КиберЛенинка http://cyberleninka.ru/;
4 Американская патентная база данных http://www.uspto.gov/patft/
5 Министерство науки и высшего образования Российской Федерации https://www.minobrnauki.gov.ru/;
6 Федеральный портал "Российское образование" http://www.edu.ru/;
7 Информационная система "Единое окно доступа к образовательным ресурсам" http://window.edu.ru/;
8 Единая коллекция цифровых образовательных ресурсов http://school-collection.edu.ru/ .
9 Проект Государственного института русского языка имени А.С. Пушкина "Образование на русском" https://pushkininstitute.ru/;
10 Справочно-информационный портал "Русский язык" http://gramota.ru/;
11 Служба тематических толковых словарей http://www.glossary.ru/;
12 Словари и энциклопедии http://dic.academic.ru/;
13 Образовательный портал "Учеба" http://www.ucheba.com/;
14 Законопроект "Об образовании в Российской Федерации". Вопросы и ответы http://xn--273--84d1f.xn--p1ai/voprosy_i_otvety

Собственные электронные образовательные и информационные ресурсы КубГУ

5 Электронный каталог Научной библиотеки КубГУ http://megapro.kubsu.ru/MegaPro/Web
6 Электронная библиотека трудов ученых КубГУ http://megapro.kubsu.ru/MegaPro/UserEntry?Action=ToDb&idb=6
7 Среда модульного динамического обучения http://moodle.kubsu.ru
8 База учебных планов, учебно-методических комплексов, публикаций и конференций http://infoneeds.kubsu.ru/
9 Библиотека информационных ресурсов кафедры информационных образовательных технологий http://mschool.kubsu.ru;
10 Электронный архив документов КубГУ http://docspace.kubsu.ru/

11 Электронные образовательные ресурсы кафедры информационных систем и технологий в образовании КубГУ и научно-методического журнала "ШКОЛЬНЫЕ ГОДЫ" http://icdau.kubsu.ru/

5.5 Перечень информационно-коммуникационных технологий

1. Компьютерное тестирование представленных программ.
2. Использование электронных презентаций при проведении лекционных занятий.

5.6 Перечень лицензионного и свободно распространяемого программного обеспечения

- SWI-Prolog;
- Java;
- Clojure;
- Scala.

6. Методические указания для обучающихся по освоению дисциплины (модуля)

Все задания выполняются в Clojure, Scala, SWI-Prolog в любой операционной системе.

Программное обеспечение:

- SWI-Prolog;
- Java;
- Clojure;
- Scala.

В освоении дисциплины инвалидами и лицами с ограниченными возможностями здоровья большое значение имеет индивидуальная учебная работа (консультации) – дополнительное разъяснение учебного материала.

Индивидуальные консультации по предмету являются важным фактором, способствующим индивидуализации обучения и установлению воспитательного контакта между преподавателем и обучающимся инвалидом или лицом с ограниченными возможностями здоровья.

7. Материально-техническое обеспечение по дисциплине (модулю)

№	Вид работ	Наименование учебной аудитории, ее оснащенность оборудованием и техническими средствами обучения
1.	Лекционные занятия	Аудитория, укомплектованная специализированной мебелью и техническими средствами обучения
2.	Лабораторные занятия	Аудитория, укомплектованная специализированной мебелью и техническими средствами обучения, компьютерами, проектором, программным обеспечением
3.	Практические занятия	Аудитория, укомплектованная специализированной мебелью и техническими средствами обучения
4.	Групповые (индивидуальные) консультации	Аудитория, укомплектованная специализированной мебелью и техническими средствами обучения, компьютерами, программным обеспечением
5.	Текущий контроль, промежуточная аттестация	Аудитория, укомплектованная специализированной мебелью и техническими средствами обучения, компьютерами, программным обеспечением
6.	Самостоятельная работа	Кабинет для самостоятельной работы, оснащенный компьютерной техникой с возможностью подключения к

		сети «Интернет»,программой экранного увеличения и обеспеченный доступом в электронную информационно-образовательную среду университета.

Примечание: Конткретизация аудиторий и их оснащение определяется ОПОП.