МОСКОВСКИЙ УНИВЕРСИТЕТ ИМЕНИ С.Ю. ВИТТЕ

Рабочая программа дисциплины

Высокоуровневые методы программирования

Направление подготовки:
09.03.03 «Прикладная информатика»

Направленность (профиль) программы:
Искусственный интеллект и анализ данных

Уровень высшего образования:
бакалавриат

Шифр модуля	Наименование модуля
09.03.03:ВысМетПрог:1	Высокоуровневые методы программирования1 / Объектно-ориентированное программирование
09.03.03:ВысМетПрог:2	Высокоуровневые методы программирования2 / Функциональное программирование
09.03.03:ВысМетПрог:3	Высокоуровневые методы программирования3 / Создание графического интерфейса пользователя
09.03.03:ВысМетПрог:4	Высокоуровневые методы программирования4 / Web-программирование
09.03.03:ВысМетПрог:5	Высокоуровневые методы программирования5 / Программирование сетевых приложений
09.03.03:ВысМетПрог:6	Высокоуровневые методы программирования6 / Многопоточное программирование
09.03.03:ВысМетПрог:7	Высокоуровневые методы программирования7 / Программирование запросов к базе данных из Python
09.03.03:ВысМетПрог:8	Высокоуровневые методы программирования8 / Расширенные возможности работы с файлами

СОДЕРЖАНИЕ

1. Цели и задачи дисциплины

Цель - овладеть высокоуровневыми методами программирования на языке Python с целью успешной интеграции в сферу программирования в игровой компьютерной индустрии.

Задачи:

- изучение основных концепций и синтаксиса высокоуровневого языка программирования Python, необходимых для разработки игровых приложений;
- освоение принципов объектно-ориентированного программирования (ООП) на Python и их применение в создании игровых компонентов и систем;
- разработка навыков использования библиотек и фреймворков, специфичных для игровой индустрии, на языке Python;
- активное участие в проектах по созданию игровых приложений, направленных на применение высокоуровневых методов программирования на Python;
- изучение паттернов проектирования и их использование в контексте разработки игровых систем и интерфейсов с применением Python;
- овладение методами оптимизации и управления ресурсами в игровых приложениях с использованием высокоуровневых подходов программирования;
- разработка навыков создания многопользовательских и онлайн-игровых приложений с использованием высокоуровневого языка программирования Python;
- применение высокоуровневых методов программирования на Python для решения специфических задач, связанных с созданием игровых механик и искусственного интеллекта в играх.

2. Перечень планируемых результатов обучения по дисциплине, соотнесенных с планируемыми результатами освоения образовательной программы

2.1. Компетенции обучающегося, формируемые в результате освоения дисциплины.

В результате освоения дисциплины у обучающихся должны быть сформированы компетенции в соответствии с ФГОС ВО по направлению подготовки и с учетом обобщенных трудовых функции и трудовых функций профессиональных стандартов 06.001 «ПРОГРАММИСТ», 06.015 «СПЕЦИАЛИСТ ПО ИНФОРМАЦИОННЫМ СИСТЕМАМ», 06.042 «СПЕЦИАЛИСТ ПО БОЛЬШИМ ДАННЫМ» к выполнению которых в ходе обучения готовится обучающийся.

Соотношение обобщённых трудовых функций (ОТФ) и трудовых функций, имеющих отношение к будущей профессиональной деятельности обучающегося (ТФ):

Код и наименование профессионального стандарта	Код и наименование ОТФ	Код и наименование ТФ
06.001 ПРОГРАММИСТ	D Разработка требований и проектирование программного обеспечения	D/03.6 Проектирование компьютерного программного обеспечения
06.015 СПЕЦИАЛИСТ ПО ИНФОРМАЦИОННЫМ СИСТЕМАМ	C Выполнение работ и управление работами по созданию (модификации) и сопровождению ИС,	C/16.6 Проектирование и дизайн ИС в рамках выполнения работ и управления работами по

	автоматизирующих задачи организационного управления и бизнес-процессы	созданию (модификации) и сопровождению ИС
06.042 СПЕЦИАЛИСТ ПО БОЛЬШИМ ДАННЫМ	А Анализ больших данных с использованием существующей в организации методологической и технологической инфраструктуры	А/01.6 Выявление, формирование и согласование требований к результатам аналитических работ с применением технологий больших данных А/03.6 Подготовка данных для проведения аналитических работ по исследованию больших данных

Процесс изучения дисциплины направлен на формирование у обучающихся следующих компетенций (результатов освоения образовательной программы):

Коды компетенций	Содержание компетенций
ПК-1	Способность разработки прикладного программного обеспечения, автоматизации работы с базами данных и документами, программирования бизнес-логики приложений, интеграции разнородных данных

2.2. Взаимосвязь планируемых результатов обучения по дисциплине с формируемыми компетенциями ОПОП

Коды компетенций ОПОП	Индикаторы	Знать	Уметь	Владеть
ПК-1	ПК-1.1. Знает технологии программирования прикладного программного обеспечения и бизнес-логики приложений	- технологии программирования с использованием языков высокого уровня; - технологии распределенных вычислений	- использовать языки высокого уровня для разработки прикладного программного обеспечения	- методами решения типовых алгоритмических задач
	ПК-1.2. Умеет разрабатывать и конфигурировать прикладное программное обеспечение	- последовательность разработки прикладного программного обеспечения	- использовать языки программирования высокого уровня для разработки многопоточных приложений	- методами передачи данных между потоками
	ПК-1.3. Владеет навыками автоматизации решения типовых задач, работы с	- методы решения типовых алгоритмических задач; - методы проектирования	- программировать запросы к базе данных; - программировать динамически	- методами доступа к данным с помощью стандартных

	базами данных и документами, интеграции разнородных данных в корпоративных информационных системах	интерфейса на языке Python с использованием библиотек визуализации	создаваемый и обновляемый интерфейс на языке Python	интерфейсов подключения к данным; - навыками доступа к данным с помощью стандартных интерфейсов подключения

3. Место дисциплины в структуре образовательной программы

3.1. Дисциплина «Высокоуровневые методы программирования» относится к части, формируемой участниками образовательных отношений учебного плана ОПОП ВО по направлению подготовки 09.03.03 «Прикладная информатика».

4. Объем дисциплины

Общая трудоемкость (объем) дисциплины составляет 8 зач. ед. 289 часов.

Объём дисциплины	Всего часов		
	очная форма обучения	заочная форма обучения	заочная «выходного дня» форма обучения
Общая трудоемкость дисциплины	289	289	289
Контактная работа обучающегося с преподавателем (по видам учебных занятий) (всего), в том числе:	121	29	29
Занятия лекционного типа	30	8	8
Занятия семинарского типа	90	20	20
Самостоятельная работа под руководством преподавателя	0	0	0
Курсовая работа	1	1	1
Консультации	2	2	2
Контрольные часы на аттестацию, аттестация	0,7	0,7	0,7
Самостоятельная работа обучающихся СРС/подготовка к экзамену (зачету) в соответствии с БУП.	128,3/36	243,3/13	243,3/13
Формы промежуточной аттестации обучающегося (экзамен/зачет)	зачет; экзамен	зачет; экзамен	зачет; экзамен

5. Содержание дисциплины

5.1. Содержание дисциплины, структурированное по модулям и темам

№ темы	Наименование темы	Содержание темы
Модуль 1. Объектно-ориентированное программирование		
1	Основные понятия и принципы объектно-ориентированного программирования	Абстракция и декомпозиция. Объекты. Типы и классы. Инкапсуляция. Полиморфизм.
2	Отношения между классами	Отношения между классами. Наследование. Агрегация. Ассоциация. Методы классов.
Рубежный контроль (РК 1)		Модульное тестирование
Модуль 2. Функциональное программирование		

3	Модули и встроенные функции	Понятие модуля. Основные стандартные модули Python. Модули расширения. Подключение модуля к программе на Python. Встроенные функции.
4	Модули стандартной библиотеки	Сервисы периода выполнения. Поддержка цикла разработки. Взаимодействие с операционной системой (файлы, процессы). Хранение данных. Архивация. Платформо-зависимые модули: модули для UNIX и модули для Windows. Поддержка сети. Протоколы Интернет. Поддержка Internet. Форматы данных.
5	Функциональная программа	Программы в функциональном стиле. Композиция функций. Функция: определение и вызов. Список формальных параметров и тело определения функции. Функция одного аргумента. Функция двух аргументов. Функция с одним обязательным аргументом, с одним, имеющим значение по умолчанию и неопределенным числом именованных аргументов. Функция произвольного числа аргументов. Функция с обычными (позиционными) и именованными аргументами. Обработка последовательностей. Списковые включения.
6	Итераторы и простые генераторы	Объекты, обеспечивающие последовательный доступ к данным контейнера. Схемы обработки данных с помощью итераторов в модуле itertools. Функции модуля itertools. Простые генераторы. Генераторное выражение. Карринг.
Рубежный контроль (РК 2)		Модульное тестирование
Модуль 3. Создание графического интерфейса пользователя		
7	Создание графического интерфейса в библиотеке Tk	Обзор графических библиотек. Графический интерфейс в библиотеке Tk. Виджеты и их свойства. Менеджер геометрии Pack. Менеджер геометрии Place. Менеджер геометрии Grid.
8	Создание графических приложений и компьютерных игр в Tk	Text – многострочное текстовое поле. Метод bind(). События. Создание графических примитивов (рисование). Canvas. Идентификаторы, теги и анимация.
Рубежный контроль (РК 3)		Модульное тестирование
Модуль 4. Web-программирование		
9	CGI-сценарии	CGI-сценарии. Методы передачи данных из заполненной в браузере формы Web-серверу. Установленные Web-сервером переменные окружения.
10	Среды разработки web для Python	Zope и его объектная модель. Создания сервера web-приложений Zope. Документы Zope. Модули для web-сервера mod_python. Flask / Django Framework. Установка Django в Windows. Создание первого приложения. База данных и модели. Панель администрирования. Отображение данных на сайте. Расширение шаблонов. Страница объекта. Создание заявки. Фильтрация данных. Сортировка данных. Подключение дизайна. Публикация сайта в интернете.

Рубежный контроль (РК 4)		Модульное тестирование
Итоговый контроль (ПА)		зачет
Модуль 5. Программирование сетевых приложений		
11	Клиент-серверное приложение	Работа с сокетами. SMTP-соединение с сервером электронной почты. Прием почты из почтового ящика на сервере POP3.
12	Доступ к объектам WWW	Модули для клиента WWW. Функции для загрузки сетевых объектов. Функции для анализа URL. Возможности модуля urllib2. XML-RPC сервер. Протокол XML-RPC.
Рубежный контроль (РК 5)		Модульное тестирование
Модуль 6. Многопоточное программирование		
13	Основы многопоточности в Python	Виды многопоточности. Запуск и завершение потоков. Потоки управления. Состояния потоков. Приоритеты, блокировки. Конструктор класса threading.Thread. Управление жизнью потоков. Таймер
14	Передача данных между потоками	Передача данных между потоками. Управление доступом к потокам. Замки. Тупиковая ситуация (deadlock). Семафоры. События. Условия. Очередь. Визуализация работы потоков.
Рубежный контроль (РК 6)		Модульное тестирование
Модуль 7. Программирование запросов к базе данных из Python		
15	Интерфейсы модуля расширения для работы с базой данных	Описание DB API 2.0. Интерфейс модуля. Доступ к базе данных. Объект-соединение. Объект-курсор. Объекты-типы.
16	Работа с базой данных из Python-приложения	Работа с базой данных из Python-приложения. Модуль sqlite. Создание базы данных. Наполнение базы данных. Выборки из базы данных. Модули, поддерживающие DB-API 2.0 для MySQL, Oracle, Microsoft ActiveX Data Objects, Sybase.
Рубежный контроль (РК 7)		Модульное тестирование
Модуль 8. Расширенные возможности работы с файлами		
17	Работа с файлами и сообщениями	Работа с файлами. Работа с табличными данными. Формат CSV. Пакет email. Разбор сообщения. Формирование сообщения. Разбор поля заголовка.
18	Работа с XML файлами	XML файлы. Формирование XML-документа. Анализ XML-документа. Пространства имен.
Рубежный контроль (РК 8)		Модульное тестирование
Курсовая работа		Курсовая работа
Итоговый контроль (ПА)		экзамен

* для обучающихся по заочной форме обучения

5.2. Модули и темы дисциплины, их трудоемкость по видам учебных занятий

Очная форма обучения

№ те-мы	Модули и темы дисциплины	Все-го	Виды учебной работы, включая самостоятельную работу студентов и трудоемкость в т.ч.						Процедура оценивания/ оцениваемые компетенции
			ЛЗ	СЗ	ЛР	СРС	КАтт	Конс	
Модуль 1. Объектно-ориентированное программирование		36	4	12	0	20	0	0	Текущий контроль
1.	Основные понятия и принципы объектно-ориентированного программирования	4	2	-	-	2	-	-	*Текущий опрос ($R_{спз}$); РК - Тестирование (решение ТОЗ) ПК-1*
СЗ	*Принципы объектно-ориентированного программирования*	12	-	6	-	6	-	-	
2.	Отношения между классами	8	2	-	-	6	-	-	
СЗ	*Отношения между классами*	12	-	6	-	6	-	-	
Модуль 2. Функциональное программирование		36	8	20	0	8	0	0	Текущий контроль
3.	Модули и встроенные функции	3	2	-	-	1	-	-	*Текущий опрос ($R_{спз}$); РК - Тестирование (решение ТОЗ) ПК-1*
СЗ	*Модули и встроенные функции*	5	-	4	-	1	-	-	
4.	Модули стандартной библиотеки	3	2	-	-	1	-	-	
СЗ	*Модули стандартной библиотеки*	5	-	4	-	1	-	-	
5.	Функциональная программа	3	2	-	-	1	-	-	
СЗ	*Функциональная программа*	7	-	6	-	1	-	-	
6.	Итераторы и простые генераторы	3	2	-	-	1	-	-	
СЗ	*Итераторы и простые генераторы*	7	-	6	-	1	-	-	
Модуль 3. Создание графического интерфейса пользователя		36	4	12	0	20	0	0	Текущий контроль
7.	Создание графического интерфейса в библиотеке Tk	4	2	-	-	2	-	-	*Текущий опрос ($R_{спз}$); РК - Тестирование (решение ТОЗ) ПК-1*
СЗ	*Создание графического интерфейса в библиотеке Tk*	12	-	6	-	6	-	-	
8.	Создание графических приложений и компьютерных игр в Tk	6	2	-	-	4	-	-	
СЗ	*Создание графических приложений и компьютерных игр в Tk*	14	-	6	-	8	-	-	
Модуль 4. Web-программирование		36	2	10	0	23,8	0,2	0	Текущий контроль
9.	CGI-сценарии	7	1	-	-	6	-	-	

СЗ	*Реализация web-приложения при использовании CGI-сценариев*	11	-	4	-	7	-	-	*Текущий опрос ($R_{спз}$); РК - Тестирование (решение ТОЗ) ПК-1*
10.	Среды разработки web для Python	5	1	-	-	4	-	-	
СЗ	*Среды разработки web-приложений для Python*	12,8	-	6	-	6,8	-	-	
Контрольные часы на аттестацию, аттестация		0,2	-	-	-	-	0,2	-	
Итого за семестр		144	18	54	0	71,8	0,2	0	
Модуль 5. Программирование сетевых приложений		36	3	12	0	21	0	0	Текущий контроль
11.	Клиент-серверное приложение	7	1	-	-	6	-	-	*Текущий опрос ($R_{спз}$); РК - Тестирование (решение ТОЗ) ПК-1*
СЗ	*Сетевые приложения на Python*	11	-	6	-	5	-	-	
12.	Доступ к объектам WWW	6	2	-	-	4	-	-	
СЗ	*Модули для клиента WWW*	12	-	6	-	6	-	-	
Модуль 6. Многопоточное программирование		36	3	8	0	25	0	0	Текущий контроль
13.	Основы многопоточности в Python	7	1	-	-	6	-	-	*Текущий опрос ($R_{спз}$); РК - Тестирование (решение ТОЗ) ПК-1*
СЗ	*Многопоточное программирование*	9	-	4	-	5	-	-	
14.	Передача данных между потоками	8	2	-	-	6	-	-	
СЗ	*Управление потоками*	12	-	4	-	8	-	-	
Модуль 7. Программирование запросов к базе данных из Python		36	2	8	0	26	0	0	Текущий контроль
15.	Интерфейсы модуля расширения для работы с базой данных	8	1	-	-	7	-	-	*Текущий опрос ($R_{спз}$); РК - Тестирование (решение ТОЗ) ПК-1*
СЗ	*Модуль DB API 2.0*	11	-	4	-	7	-	-	
16.	Работа с базой данных из Python-приложения	7	1	-	-	6	-	-	
СЗ	*Работа с базой данных из Python-приложения*	10	-	4	-	6	-	-	
Модуль 8. Расширенные возможности работы с файлами		36	4	8	0	21,5	0,5	2	Текущий контроль
17.	Работа с файлами и сообщениями	8	2	-	-	6	-	-	*Текущий опрос ($R_{спз}$); РК - Тестирование (решение ТОЗ) ПК-1*
СЗ	*Работа с файлами и сообщениями*	8	-	4	-	4	-	-	
18.	Работа с XML файлами	8	2	-	-	6	-	-	
СЗ	*Язык XML*	9,5	-	4	-	5,5	-	-	
Контрольные часы на аттестацию, аттестация		2,5	-	-	-	-	0,5	2	
в том числе курсовая работа (КР)		1	-	-	-	1	-	-	
Итого за семестр		144	12	36	0	93,5	0,5	2	

Общий объем трудоемкости (учебной нагрузки) в часах	288	30	90	0	165,3 (36 ч. на экз)	0,7	2	

ЛЗ – занятия лекционного типа
ЛР – лабораторные работы
СЗ – занятия семинарского типа
СР – самостоятельная работа
КАтт – контрольные часы на аттестацию, аттестация
Конс – консультации

Заочная форма обучения

№ те-мы	Модули и темы дисциплины	Все-го	Виды учебной работы, включая самостоятельную работу студентов и трудоемкость в т.ч.						Процедура оценивания/ оцениваемые компетенции
			ЛЗ	СЗ	ЛР	СРС	КАтт	Конс	
Модуль 1. Объектно-ориентированное программирование		36	1	2	0	33	0	0	Текущий контроль
1.	Основные понятия и принципы объектно-ориентированного программирования	9	1	-	-	8	-	-	*Текущий опрос ($R_{спз}$); РК - Тестирование (решение ТОЗ) ПК-1*
СЗ	*Принципы объектно-ориентированного программирования*	9	-	1	-	8	-	-	
2.	Отношения между классами	9	-	-	-	9	-	-	
СЗ	*Отношения между классами*	9	-	1	-	8	-	-	
Модуль 2. Функциональное программирование		36	1	2	0	33	0	0	Текущий контроль
3.	Модули и встроенные функции	5	-	-	-	5	-	-	*Текущий опрос ($R_{спз}$); РК - Тестирование (решение ТОЗ) ПК-1*
СЗ	*Модули и встроенные функции*	4	-	-	-	4	-	-	
4.	Модули стандартной библиотеки	4	-	-	-	4	-	-	
СЗ	*Модули стандартной библиотеки*	4	-	-	-	4	-	-	
5.	Функциональная программа	5	1	-	-	4	-	-	
СЗ	*Функциональная программа*	6	-	2	-	4	-	-	
6.	Итераторы и простые генераторы	4	-	-	-	4	-	-	
СЗ	*Итераторы и простые генераторы*	4	-	-	-	4	-	-	
Модуль 3. Создание графического интерфейса пользователя		36	1	2	0	33	0	0	Текущий контроль
7.	Создание графического интерфейса в библиотеке Tk	10	1	-	-	9	-	-	*Текущий опрос ($R_{спз}$); РК -*

СЗ	*Создание графического интерфейса в библиотеке Tk*	8	-	-	-	8	-	-	*Тестирование (решение ТОЗ) ПК-1*
8.	Создание графических приложений и компьютерных игр в Tk	8	-	-	-	8	-	-	
СЗ	*Создание графических приложений и компьютерных игр в Tk*	10	-	2	-	8	-	-	
Модуль 4. Web-программирование		36	1	4	0	30,8	0,2	0	Текущий контроль
9.	CGI-сценарии	8	1	-	-	7	-	-	*Текущий опрос ($R_{спз}$); РК - Тестирование (решение ТОЗ) ПК-1*
СЗ	*Реализация web-приложения при использовании CGI-сценариев*	10	-	2	-	8	-	-	
10.	Среды разработки web для Python	7	-	-	-	7	-	-	
СЗ	*Среды разработки web-приложений для Python*	10,8	-	2	-	8,8	-	-	
Контрольные часы на аттестацию, аттестация		0,2	-	-	-	-	0,2	-	
В т.ч. выполнение рейтинговой работы (Расчетно-аналитическое задание (БРС))									*Выполнение Расчетно-аналитическое задание (БРС) / ПК-1*
Итого за семестр		144	4	10	0	129,8	0,2	0	
Модуль 5. Программирование сетевых приложений		36	1	2	0	33	0	0	Текущий контроль
11.	Клиент-серверное приложение	10	1	-	-	9	-	-	*Текущий опрос ($R_{спз}$); РК - Тестирование (решение ТОЗ) ПК-1*
СЗ	*Сетевые приложения на Python*	9	-	1	-	8	-	-	
12.	Доступ к объектам WWW	8	-	-	-	8	-	-	
СЗ	*Модули для клиента WWW*	9	-	1	-	8	-	-	
Модуль 6. Многопоточное программирование		36	1	4	0	31	0	0	Текущий контроль
13.	Основы многопоточности в Python	9	1	-	-	8	-	-	*Текущий опрос ($R_{спз}$); РК - Тестирование (решение ТОЗ) ПК-1*
СЗ	*Многопоточное программирование*	10	-	2	-	8	-	-	
14.	Передача данных между потоками	8	-	-	-	8	-	-	
СЗ	*Управление потоками*	9	-	2	-	7	-	-	
Модуль 7. Программирование запросов к базе данных из Python		36	1	2	0	33	0	0	Текущий контроль
15.	Интерфейсы модуля расширения для работы с базой данных	10	1	-	-	9	-	-	*Текущий опрос ($R_{спз}$); РК - Тестирование*
СЗ	*Модуль DB API 2.0*	9	-	1	-	8	-	-	

16.	Работа с базой данных из Python-приложения	8	-	-	-	8	-	-	*(решение ТОЗ) ПК-1*
СЗ	*Работа с базой данных из Python-приложения*	9	-	1	-	8	-	-	
Модуль 8. Расширенные возможности работы с файлами		36	1	2	0	30,5	0,5	2	Текущий контроль
17.	Работа с файлами и сообщениями	9	1	-	-	8	-	-	*Текущий опрос ($R_{спз}$); РК - Тестирование (решение ТОЗ) ПК-1*
СЗ	*Работа с файлами и сообщениями*	8	-	-	-	8	-	-	
18.	Работа с XML файлами	7	-	-	-	7	-	-	
СЗ	*Язык XML*	9,5	-	2	-	7,5	-	-	
Контрольные часы на аттестацию, аттестация		2,5	-	-	-	-	0,5	2	
в том числе курсовая работа (КР)		1	-	-	-	1	-	-	
Итого за семестр		144	4	10	0	127,5	0,5	2	
Общий объем трудоемкости (учебной нагрузки) в часах		288	8	20	0	257,3 (13 ч. на экз)	0,7	2	

Заочная, выходного дня форма обучения

№ те-мы	Модули и темы дисциплины	Все-го	Виды учебной работы, включая самостоятельную работу студентов и трудоемкость в т.ч.						Процедура оценивания/ оцениваемые компетенции
			ЛЗ	СЗ	ЛР	СРС	КАтт	Конс	
Модуль 1. Объектно-ориентированное программирование		36	1	3	0	32	0	0	Текущий контроль
1.	Основные понятия и принципы объектно-ориентированного программирования	9	1	-	-	8	-	-	*Текущий опрос ($R_{спз}$); РК - Тестирование (решение ТОЗ) ПК-1*
СЗ	*Принципы объектно-ориентированного программирования*	9	-	1	-	8	-	-	
2.	Отношения между классами	8	-	-	-	8	-	-	
СЗ	*Отношения между классами*	10	-	2	-	8	-	-	
Модуль 2. Функциональное программирование		36	1	2	0	33	0	0	Текущий контроль
3.	Модули и встроенные функции	5	-	-	-	5	-	-	*Текущий опрос ($R_{спз}$); РК - Тестирование (решение ТОЗ) ПК-1*
СЗ	*Модули и встроенные функции*	4	-	-	-	4	-	-	
4.	Модули стандартной библиотеки	4	-	-	-	4	-	-	
СЗ	*Модули стандартной библиотеки*	4	-	-	-	4	-	-	

5.	Функциональная программа	5	1	-	-	4	-	-	
СЗ	*Функциональная программа*	6	-	2	-	4	-	-	
6.	Итераторы и простые генераторы	4	-	-	-	4	-	-	
СЗ	*Итераторы и простые генераторы*	4	-	-	-	4	-	-	
Модуль 3. Создание графического интерфейса пользователя		36	1	3	0	32	0	0	Текущий контроль
7.	Создание графического интерфейса в библиотеке Tk	9	1	-	-	8	-	-	*Текущий опрос ($R_{спз}$); РК - Тестирование (решение ТОЗ) ПК-1*
СЗ	*Создание графического интерфейса в библиотеке Tk*	9	-	1	-	8	-	-	
8.	Создание графических приложений и компьютерных игр в Tk	8	-	-	-	8	-	-	
СЗ	*Создание графических приложений и компьютерных игр в Tk*	10	-	2	-	8	-	-	
Модуль 4. Web-программирование		36	1	2	0	32,8	0,2	0	Текущий контроль
9.	CGI-сценарии	10	1	-	-	9	-	-	*Текущий опрос ($R_{спз}$); РК - Тестирование (решение ТОЗ) ПК-1*
СЗ	*Реализация web-приложения при использовании CGI-сценариев*	10	-	1	-	9	-	-	
10.	Среды разработки web для Python	7	-	-	-	7	-	-	
СЗ	*Среды разработки web-приложений для Python*	8,8	-	1	-	7,8	-	-	
Контрольные часы на аттестацию, аттестация		0,2	-	-	-	-	0,2	-	
В т.ч. выполнение рейтинговой работы (Расчетно-аналитическое задание (БРС))									*Выполнение Расчетно-аналитическое задание (БРС) / ПК-1*
Итого за семестр		144	4	10	0	129,8	0,2	0	
Модуль 5. Программирование сетевых приложений		36	1	2	0	33	0	0	Текущий контроль
11.	Клиент-серверное приложение	9	1	-	-	8	-	-	*Текущий опрос ($R_{спз}$); РК - Тестирование (решение ТОЗ) ПК-1*
СЗ	*Сетевые приложения на Python*	9	-	1	-	8	-	-	
12.	Доступ к объектам WWW	9	-	-	-	9	-	-	
СЗ	*Модули для клиента WWW*	9	-	1	-	8	-	-	
Модуль 6. Многопоточное программирование		36	1	2	0	33	0	0	Текущий контроль
13.	Основы многопоточности в Python	9	1	-	-	8	-	-	

СЗ	*Многопоточное программирование*	9	-	1	-	8	-	-	*Текущий опрос ($R_{спз}$); РК - Тестирование (решение ТОЗ) ПК-1*
14.	Передача данных между потоками	9	-	-	-	9	-	-	
СЗ	*Управление потоками*	9	-	1	-	8	-	-	
Модуль 7. Программирование запросов к базе данных из Python		36	1	3	0	32	0	0	Текущий контроль
15.	Интерфейсы модуля расширения для работы с базой данных	9	1	-	-	8	-	-	*Текущий опрос ($R_{спз}$); РК - Тестирование (решение ТОЗ) ПК-1*
СЗ	*Модуль DB API 2.0*	10	-	2	-	8	-	-	
16.	Работа с базой данных из Python-приложения	8	-	-	-	8	-	-	
СЗ	*Работа с базой данных из Python-приложения*	9	-	1	-	8	-	-	
Модуль 8. Расширенные возможности работы с файлами		36	1	3	0	29,5	0,5	2	Текущий контроль
17.	Работа с файлами и сообщениями	7	1	-	-	6	-	-	*Текущий опрос ($R_{спз}$); РК - Тестирование (решение ТОЗ) ПК-1*
СЗ	*Работа с файлами и сообщениями*	9	-	2	-	7	-	-	
18.	Работа с XML файлами	8	-	-	-	8	-	-	
СЗ	*Язык XML*	9,5	-	1	-	8,5	-	-	
Контрольные часы на аттестацию, аттестация		2,5	-	-	-	-	0,5	2	
в том числе курсовая работа (КР)		1	-	-	-	1	-	-	
Итого за семестр		144	4	10	0	127,5	0,5	2	
Общий объем трудоемкости (учебной нагрузки) в часах		288	8	20	0	257,3 (13 ч. на экз)	0,7	2	

6. Учебно-методическое обеспечение самостоятельной работы обучающихся

6.1.Задания для самостоятельной подготовки к занятиям семинарского типа

Семинарское занятие по теме 1

Тема: Принципы объектно-ориентированного программирования

Цель: изучить принципы объектно-ориентированного программирования в Python; изучить синтаксис и применение встроенных функций в Python.

Задания (вопросы) для подготовки:

В языке Python для определения класса используется оператор class:

```
class имя_класса(класс1, класс2, ...):
  # определения методов
```

Объект-класс, как и объект-функция, может быть вызван. Это и будет вызовом конструктора:

```
>>> import sets
>>> s = sets.Set([1, 2, 3])
```

Пример определения класса:

```
from sets import Set as set  # тип для множества

class G:
  def __init__(self, V, E):
    self.vertices = set(V)
    self.edges = set(E)

  def add_vertex(self, v):
    self.vertices.add(v)

  def add_edge(self, (v1, v2)):
    self.vertices.add(v1)
    self.vertices.add(v2)
    self.edges.add((v1, v2))

  def has_edge(self, (v1, v2)):
    return (v1, v2) in self.edges

  def __str__(self):
    return "%s; %s" % (self.vertices, self.edges)
```

Использовать класс можно следующим образом:

g = G([1, 2, 3, 4], [(1, 2), (2, 3), (2, 4)]) print g g.add_vertex(5) g.add_edge((5,6)) print g.has_edge((1,6)) print g

что даст в результате

Set([1, 2, 3, 4]); Set([(2, 4), (1, 2), (2, 3)]) False Set([1, 2, 3, 4, 5, 6]); Set([(2, 4), (1, 2), (5, 6), (2, 3)])

Доступ к свойствами:

```
class C(object):
  def getx(self): return self.__x
  def setx(self, value): self.__x = value
  def delx(self): del self.__x
  x = property(getx, setx, delx, "I'm the 'x' property.")
```

Синтаксически доступ к свойству x будет обычной ссылкой на атрибут:

>>> c = C() >>> c.x = 1 >>> print c.x 1 >>> del c.x

Пример доступа к любым атрибутам экземпляра класса

из словаря создается объект, именами атрибутов которого будут ключи словаря, а значениями - значения из словаря по заданным ключам

```
class AttDict(object):
  def __init__(self, dict=None):
    object.__setattr__(self, '_selfdict', dict or {})

  def __getattr__(self, name):
    if self._selfdict.has_key(name):
      return self._selfdict[name]
    else:
      raise AttributeError

  def __setattr__(self, name, value):
    if name[0] != '_':
      self._selfdict[name] = value
    else:
      raise AttributeError

  def __delattr__(self, name):
    if name[0] != '_' and self._selfdict.has_key(name):
      del self._selfdict[name]

ad = AttDict({'a': 1, 'b': 10, 'c': '123'})
print ad.a, ad.b, ad.c
ad.d = 512
print ad.d
```

Сокрытие данных

Двойное подчеркивание работает как указание на то, что атрибут - приватный. При этом атрибут все же доступен, но уже под другим именем, что и иллюстрируется ниже:

```
 >>> class X:
...   x = 0
...   _x = 0
...   __x = 0
...
>>> dir(X)
['_X__x', '__doc__', '__module__', '_x', 'x']
```

Полиморфизм в Python:

def get_last(x): return x[-1] print get_last([1, 2, 3]) print get_last("abcd")

Управление сравнением экземпляров класса:

```
class Point:
  def __init__(self, x, y):
    self.coord = (x, y)
  def __nonzero__(self):
    return self.coord[0] != 0 or self.coord[1] != 0
  def __cmp__(self, p):
    return cmp(self.coord, p.coord)

for x in range(-3, 4):
  for y in range(-3, 4):
    if Point(x, y) < Point(y, x):
      print "*",
    elif Point(x, y):
      print ".",
    else:
      print "o",
  print
```

Программа выведет:

```
. * * * * * *
. . * * * * *
. . . * * * *
. . . o * * *
. . . . . * *
. . . . . . *
. . . . . . .
```

В данной программе класс Point (Точка) имеет метод __nonzero__(), который определяет истинностное значение объекта класса. Истину будут давать только точки, отличные от (0, 0). Другой метод - __cmp__() - вызывается при необходимости сравнить точку и другой объект (имеющий как и точка атрибут coord, который содержит кортеж как минимум из двух элементов). Нужно заметить, что вместо __cmp__ можно определить отдельные методы для операций сравнения: __lt__, __le__, __ne__, __eq__, __ge__, __gt__ (для <, <=, !=, ==, >=, > соответственно).

Имитация числовых типов.

Класс, который пользуется удобством синтаксиса инфиксного +, можно определить так:

```
class Plussable:
  def __add__(self, x):
    ...
  def __radd__(self, x):
    ...
```

```
 def __iadd__(self, x):
   ...
```

Семинарское занятие по теме 2

Тема: Отношения между классами

Цель: изучить возможности наследования, агрегации, ассоциации и методов классов в Python; изучить модули стандартной библиотеки.

Задания (вопросы) для подготовки:

Наследование

Функция issubclass(x, y) может сказать, является ли класс x подклассом класса y:

```
 >>> class A(object): pass
...
>>> class B(A): pass
...
>>> issubclass(A, object)
True
>>> issubclass(B, A)
True
>>> issubclass(B, object)
True
>>> issubclass(A, str)
False
>>> issubclass(A, A)  # класс является подклассом самого себя
True
```

Множественное наследование

способ сбора нужных комбинаций методов в серии классов:

```
 class A:
  def a(self): return 'a'
class B:
  def b(self): return 'b'
class C:
  def c(self): return 'c'

class AB(A, B):
  pass
class BC(B, C):
  pass
class ABC(A, B, C):
  pass
```

Порядок разрешения методов

В случае, когда надклассы имеют одинаковые методы, использование того или иного метода определяется **порядком разрешения методов** (method resolution order). Для "новых" классов узнать этот порядок очень просто с помощью атрибута __mro__:

>>> str.__mro__ (<type 'str'>, <type 'basestring'>, <type 'object'>)

Это означает, что сначала методы ищутся в классе str, затем в basestring, а уже потом - в object.

В Python есть встроенные контейнеры (словаря и списка), Ниже приведен класс Стек, реализованный на базе списка:

class Stack: def __init__(self): """Инициализация стека""" self._stack = [] def top(self): """Возвратить вершину стека (не снимая)""" return self._stack[-1] def pop(self): """Снять со стека элемент""" return self._stack.pop() def push(self, x): """Поместить элемент на стек""" self._stack.append(x) def __len__(self): """Количество элементов в стеке""" return len(self._stack) def __str__(self): """Представление в виде строки""" return " : ".join(["%s" % e for e in self._stack])

Использование:

>>> s = Stack() >>> s.push(1) >>> s.push(2) >>> s.push("abc") >>> print s.pop() abc >>> print len(s) 2 >>> print s 1 : 2

Метод класса

В Python классы являются объектами. В отличие от статического метода, в метод класса первым параметром передается объект-класс. Вместо self для подчеркивания принадлежности метода к методам класса принято использовать cls.

Пример определения класса Tree (базового класса для других подклассов). Метод convert класса Tree определяет процедуру преобразования дерева одного типа в дерево другого типа. Эта процедура абстрагируется от деталей реализации конкретных типов, описывая обобщенный алгоритм преобразования:

class Tree: # ... def convert(cls, val): if isinstance(val, Tree): children = [cls.convert(child) for child in val] return cls(val.node, children) else: return val convert = classmethod(convert)

Пример использования (метод convert()):

```
>>> # Преобразовать tree в экземпляр класса Tree
>>> tree = Tree.convert(tree)
>>> # " " " " " ParentedTree
>>> tree = ParentedTree.convert(tree)
>>> # " " " " " MultiParentedTree
>>> tree = MultiParentedTree.convert(tree)
```

Семинарское занятие по теме 3
Тема: Модули и встроенные функции
Цель: изучить синтаксис и применение встроенных функций в Python.
Задания (вопросы) для подготовки:

1. Функции преобразования типов и классы: coerce, str, repr, int, list, tuple, long, float, complex, dict, super, file, bool, object.

2. Числовые и строковые функции: abs, divmod, ord, pow, len, chr, unichr, hex, oct, cmp, round, unicode.

3. Функции обработки данных: apply, map, filter, reduce, zip, range, xrange, max, min, iter, enumerate, sum.

4. Функции определения свойств: hash, id, callable, issubclass, isinstance, type.

5. Функции для доступа к внутренним структурам: locals, globals, vars, intern, dir.

6. Функции компиляции и исполнения: eval, execfile, reload, __import__, compile.

7. Функции ввода-вывода: input, raw_input, open.

8. Функции для работы с атрибутами: getattr, setattr, delattr, hasattr.

Семинарское занятие по теме 4
Тема: Модули стандартной библиотеки
Цель: изучить модули стандартной библиотеки.
Задания (вопросы) для подготовки:

1. Сервисы периода выполнения. Модули: sys, atexit, copy, traceback, math, cmath, random, time, calendar, datetime, sets, array, struct, itertools, locale, gettext.

2. Поддержка цикла разработки. Модули: pdb, hotshot, profile, unittest, pydoc. Пакеты docutils, distutils.

3. Взаимодействие с ОС (файлы, процессы). Модули: os, os.path, getopt, glob, popen2, shutil, select, signal, stat, tempfile.

4. Обработка текстов. Модули: string, re, StringIO, codecs, difflib, mmap, sgmllib, htmllib, htmlentitydefs. Пакет xml.

5. Многопоточные вычисления. Модули: threading, thread, Queue.

6. Хранение данных. Архивация. Модули: pickle, shelve, anydbm, gdbm, gzip, zlib, zipfile, bz2, csv, tarfile.

7. Платформо-зависимые модули. Для UNIX: commands, pwd, grp, fcntl, resource, termios, readline, rlcompleter. Для Windows: msvcrt, _winreg, winsound.

8. Поддержка сети. Протоколы Интернет. Модули: cgi, Cookie, urllib, urlparse, httplib, smtplib, poplib, telnetlib, socket, asyncore. Примеры серверов: SocketServer, BaseHTTPServer, xmlrpclib, asynchat.

9. Поддержка Internet. Форматы данных. Модули: quopri, uu, base64, binhex, binascii, rfc822, mimetools, MimeWriter, multifile, mailbox. Пакет email.

10. Модули: parser, symbol, token, keyword, inspect, tokenize, pyclbr, py_compile, compileall, dis, compiler.

11. Графический интерфейс. Модуль Tkinter.

Семинарское занятие по теме 5
Тема: Функциональная программа
Цель: изучить возможности функционального программирования.
Задания (вопросы) для подготовки:

Функция одного аргумента:

```
def swapcase(s):
   return s.swapcase()

print swapcase("ABC")
```

Функция двух аргументов, один из которых необязателен и имеет значение по умолчанию:

```
 def inc(n, delta=1):
    return n+delta

print inc(12)
print inc(12, 2)
```

Функция с одним обязательным аргументом, с одним, имеющим значение по умолчанию и неопределенным числом именованных аргументов:

```
 def wrap(text, width=70, **kwargs):
   from textwrap import TextWrapper
   # kwargs  - словарь с именами и значениями аргументов
   w = TextWrapper(width=width, **kwargs)
   return w.wrap(text)

print wrap("my long text ...", width=4)
```

Функция произвольного числа аргументов:

```
 def max_min(*args):
  # args - список аргументов в порядке их указания при вызове
  return max(args), min(args)

print max_min(1, 2, -1, 5, 3)
```

Функция с обычными (позиционными) и именованными аргументами:

```
 def swiss_knife(arg1, *args, **kwargs):
  print arg1
  print args
```

```
  print kwargs
  return None

print swiss_knife(1)
print swiss_knife(1, 2, 3, 4, 5)
print swiss_knife(1, 2, 3, a='abc', b='sdf')
# print swiss_knife(1, a='abc', 3, 4)  # !!! ошибка

lst = [2, 3, 4, 5]
dct = {'a': 'abc', 'b': 'sdf'}
print swiss_knife(1, *lst, **dct)
```

Пример определения функции с помощью lambda -выражения

```
func = lambda x, y: x + y
```

Функции как параметры и результат

Функция apply()

Функция apply() применяет функцию, переданную в качестве первого аргумента, к параметрам, которые переданы вторым и третьим аргументом. Эта функция в Python устарела, так как вызвать функцию можно с помощью обычного синтаксиса вызова функции. Позиционные и именованные параметры можно передать с использованием звездочек:

>>> lst = [1, 2, 3] >>> dct = {'a': 4, 'b': 5} >>> apply(max, lst) 3 >>> max(*lst) 3 >>> apply(dict, [], dct) {'a': 4, 'b': 5} >>> dict(**dct) {'a': 4, 'b': 5}

Обработка последовательностей

Функции range() и xrange()

Функция range() уже упоминалась при рассмотрении цикла for в дисциплине алгоритмизация и программирование. Эта функция принимает от одного до трех аргументов. Если аргумент всего один, она генерирует список чисел от 0 (включительно) до заданного числа (исключительно). Если аргументов два, то список начинается с числа, указанного первым аргументом. Если аргументов три - третий аргумент задает шаг

>>> print range(10) [0, 1, 2, 3, 4, 5, 6, 7, 8, 9] >>> print range(1, 10) [1, 2, 3, 4, 5, 6, 7, 8, 9] >>> print range(1, 10, 3) [1, 4, 7]
Функция xrange() - аналог range(), более предпочтительный для использования при последовательном доступе, например, в цикле for или с итераторами. Она возвращает специальный xrange -объект, который ведет себя почти как список, порождаемый range(), но не хранит в памяти все выдаваемые элементы.
Функция map()

Для применения некоторой функции ко всем элементам последовательности применяется функция map(f, *args). Первый параметр этой функции - функция, которая будет применяться ко всем элементам последовательности. Каждый следующий n+1 -й параметр должен быть последовательностью, так как каждый его элемент будет использован в качестве n -го параметра при вызове функции f(). Результатом будет список, составленный из результатов выполнения этой функции.

В следующем примере складываются значения из двух списков:

```
>>> l1 = [2, 7, 5, 3]
>>> l2 = [-2, 1, 0, 4]
>>> print map(lambda x, y: x+y, l1, l2)
[0, 8, 5, 7]
```

Функция filter()

Другой часто встречающейся операцией является фильтрование исходной последовательности в соответствии с некоторым предикатом (условием). Функция filter(f, seq) принимает два аргумента: функцию с условием и последовательность, из которой берутся значения. В результирующую последовательность попадут только те значения из исходной, для которой f() возвратит истину. Если в качестве f задано значение None, результирующая последовательность будет состоять из тех значений исходной, которые имеют истинностное значение True.

Например, в следующем фрагменте кода можно избавиться от символов, которые не являются буквами:

```
>>> filter(lambda x: x.isalpha(), 'Hi, there! I am eating an apple.')
'HithereIameatinganapple'
```

Списковые включения

Для более естественной записи обработки списков в Python 2 была внесена новинка: списковые включения. Фактически это специальный сокращенный синтаксис для вложенных циклов for и условий if, на самом низком уровне которых определенное выражение добавляется к списку, например:

all_pairs = [] for i in range(5): for j in range(5): if i <= j: all_pairs.append((i, j))

Все это можно записать в виде спискового включения так:

```
all_pairs = [(i, j) for i in range(5) for j in range(5) if i <= j]
```

Как легко заметить, списковые включения позволяют заменить map() и filter() на более удобные для прочтения конструкции.

В следующей таблице приведены эквивалентные выражения в разных формах:

В форме функции	**В форме спискового включения**
filter(f, lst)	[x for x in lst if f(x)]
filter(None, lst)	[x for x in lst if x]
map(f, lst)	[f(x) for x in lst]

Функция sum()

Получить сумму элементов можно с помощью функции sum():

```
>>> sum(range(10))
45
```

Функция reduce()

Для организации цепочечных вычислений (вычислений с накоплением результата) можно применять функцию reduce(), которая принимает три аргумента: функцию двух аргументов, последовательность и начальное значение. С помощью этой функции можно, в частности, реализовать функцию sum():

```
def sum(lst, start):
  return reduce(lambda x, y: x + y, lst, start)
```

```
Пример алгоритма, где накапливаются промежуточные результаты суммирования:
```

```
lst = range(10)
f = lambda x, y: (x[0] + y, x[1]+[x[0] + y])
print reduce(f, lst, (0, []))
```

В итоге получается:

(45, [0, 1, 3, 6, 10, 15, 21, 28, 36, 45])

Функция zip()

Эта функция возвращает список кортежей, в котором i -й кортеж содержит i -е элементы аргументов-последовательностей. Длина результирующей последовательности равна длине самой короткой из последовательностей-аргументов:

```
>>> print zip(range(5), "abcde")
[(0, 'a'), (1, 'b'), (2, 'c'), (3, 'd'), (4, 'e')]
```

Семинарское занятие по теме 6
Тема: Итераторы и простые генераторы
Цель: изучить схемы обработки данных с помощью итераторов в модуле itertools.
Задания (вопросы) для подготовки:

Функция iter()

Эта функция имеет два варианта использования. В первом она принимает всего один аргумент, который должен "уметь" предоставлять свой итератор. Во втором один из аргументов - функция без аргументов, другой - стоповое значение. Итератор вызывает указанную функцию до тех пор, пока та не возвратит стоповое значение. Второй вариант встречается много реже первого и обычно внутри метода класса, так как сложно порождать значения "на пустом месте":

Пример для самостоятельной работы

```
it1 = iter([1, 2, 3, 4, 5])

def forit(mystate=[]):
  if len(mystate) < 3:
    mystate.append(" ")
    return " "

it2 = iter(forit, None)

print [x for x in it1]
print [x for x in it2]
```

Функция enumerate()

Эта функция создает итератор, нумерующий элементы другого итератора. Результирующий итератор выдает кортежи, в которых первый элемент - номер (начиная с нуля), а второй - элемент исходной последовательности:

Пример для самостоятельной работы

```
>>> print [x for x in enumerate("abcd")]
[(0, 'a'), (1, 'b'), (2, 'c'), (3, 'd')]
```

Функция sorted()

Эта функция, появившаяся в Python 2.4, позволяет создавать итератор, выполняющий сортировку:

Пример для самостоятельной работы

```
>>> sorted('avdsdf')
['a', 'd', 'd', 'f', 's', 'v']
```

Функции модуля itertools.

Функция itertools.chain()

Функция chain() позволяет сделать итератор, состоящий из нескольких соединенных последовательно итераторов. Итераторы задаются в виде отдельных аргументов.

Пример для самостоятельной работы

```
from itertools import chain
it1 = iter([1,2,3])
it2 = iter([8,9,0])
for i in chain(it1, it2):
    print i,
```

Функция itertools.repeat()

Функция repeat() строит итератор, повторяющий некоторый объект заданное количество раз.

Пример для самостоятельной работы

```
for i in itertools.repeat(1, 4):
    print i,
```

Функция itertools.count()

Бесконечный итератор, дающий целые числа, начиная с заданного:

Пример для самостоятельной работы

```
for i in itertools.count(1):
    print i,
    if i > 100:
        break
```

Функция itertools.cycle()

Можно бесконечно повторять и некоторую последовательность (или значения другого итератора) с помощью функции cycle():

Пример для самостоятельной работы

```
tango = [1, 2, 3]
for i in itertools.cycle(tango):
    print i,
```

Функции itertools.imap(), itertools.starmap() и itertools.ifilter()

Аналогами map() и filter() в модуле itertools являются imap() и ifilter(). Отличие imap() от map() в том, что вместо значения от преждевременно завершившихся итераторов объект None не подставляется.

Пример для самостоятельной работы

```
for i in map(lambda x, y: (x,y), [1,2], [1,2,3]):
    print i,

(1, 1) (2, 2) (None, 3)

from itertools import imap
for i in imap(lambda x, y: (x,y), [1,2], [1,2,3]):
  print i,
```

Функция itertools.starmap() подобна **itertools.imap()**, но имеет всего два аргумента. Второй аргумент - последовательность кортежей, каждый кортеж которой задает набор параметров для функции (первого аргумента):

Пример для самостоятельной работы

```
>>> from itertools import starmap
>>> for i in starmap(lambda x, y: str(x) + y, [(1,'a'), (2,'b')]):
...   print i,
...
1a 2b
```

Функция **ifilter()** работает как filter(). Кроме того, в модуле itertools есть функция ifilterfalse(), которая как бы добавляет отрицание к значению функции:

Пример для самостоятельной работы

```
for i in ifilterfalse(lambda x: x > 0, [1, -2, 3, -3]):
    print i,
```

Функции itertools.takewhile() и itertools.dropwhile()

Некоторую новизну вносит другой вид фильтра: takewhile() и его "отрицательный" аналог dropwhile().

Пример для самостоятельной работы

```
for i in takewhile(lambda x: x > 0, [1, -2, 3, -3]):
    print i,

print
for i in dropwhile(lambda x: x > 0, [1, -2, 3, -3]):
  print i,
```

Функция itertools.izip()

Функция izip() аналогична встроенной zip(), но не тратит много памяти на построение списка кортежей, так как итератор выдает их строго по требованию.

Функция itertools.groupby()

Эта функция дебютировала в Python 2.4. Функция принимает два аргумента: итератор (обязательный) и необязательный аргумент - функцию, дающую значение ключа: groupby(iterable[, func]). Результатом является итератор, который возвращает двухэлементный кортеж: ключ и итератор по идущим подряд элементам с этим ключом. Если второй аргумент опущен, элемент итератора сам является ключом. В следующем примере группируются идущие подряд положительные и отрицательные элементы.

Пример для самостоятельной работы

```
import itertools, math
lst = map(lambda x: math.sin(x*.4), range(30))
for k, i in itertools.groupby(lst, lambda x: x > 0):
  print k, list(i);
```

Функция itertools.tee()

Эта функция тоже появилась в Python 2.4. Она позволяет клонировать итераторы. Первый аргумент - итератор, подлежащий клонированию. Второй (N) -- количество необходимых копий. Функция возвращает кортеж из N итераторов. По умолчанию N=2. Функция имеет смысл, только если итераторы задействованы более или менее параллельно. В противном случае выгоднее превратить исходный итератор в список.

Собственный итератор

Пример для самостоятельной работы (создание собственного итератора)

```
class Fibonacci:
  """Итератор последовательности Фибоначчи до N"""

  def __init__(self, N):
    self.n, self.a, self.b, self.max = 0, 0, 1, N

  def __iter__(self):
    # сами себе итератор: в классе есть метод next()
    return self

  def next(self):
    if self.n < self.max:
      a, self.n, self.a, self.b = self.a, self.n+1, self.b, self.a+self.b
      return a
    else:
      raise StopIteration

# Использование:
for i in Fibonacci(100):
  print i,
```

Семинарское занятие по теме 7
Тема: Создание графического интерфейса в библиотеке Tk
Цель: изучить принципы работы в библиотеке Tk.
Задания (вопросы) для подготовки:

Менеджер геометрии **grid()** в **Tkinter** можно использовать для **создания сетки кнопок для калькулятора**.

```
from tkinter import Tk, W, E

from tkinter.ttk import Frame, Button, Entry, Style



class Example(Frame):


    def __init__(self):

        super().__init__()

        self.initUI()


    def initUI(self):

        self.master.title("Калькулятор на Tkinter")


        Style().configure("TButton", padding=(0, 5, 0, 5), font='serif 10')


        self.columnconfigure(0, pad=3)

        self.columnconfigure(1, pad=3)

        self.columnconfigure(2, pad=3)

        self.columnconfigure(3, pad=3)


        self.rowconfigure(0, pad=3)

        self.rowconfigure(1, pad=3)

        self.rowconfigure(2, pad=3)

        self.rowconfigure(3, pad=3)

        self.rowconfigure(4, pad=3)
```

```
entry = Entry(self)
entry.grid(row=0, columnspan=4, sticky=W+E)
cls = Button(self, text="Очистить")
cls.grid(row=1, column=0)
bck = Button(self, text="Удалить")
bck.grid(row=1, column=1)
lbl = Button(self)
lbl.grid(row=1, column=2)
clo = Button(self, text="Закрыть")
clo.grid(row=1, column=3)
sev = Button(self, text="7")
sev.grid(row=2, column=0)
eig = Button(self, text="8")
eig.grid(row=2, column=1)
nin = Button(self, text="9")
nin.grid(row=2, column=2)
div = Button(self, text="/")
div.grid(row=2, column=3)

fou = Button(self, text="4")
fou.grid(row=3, column=0)
fiv = Button(self, text="5")
fiv.grid(row=3, column=1)
six = Button(self, text="6")
six.grid(row=3, column=2)
```

```
        mul = Button(self, text="*")
        mul.grid(row=3, column=3)

        one = Button(self, text="1")
        one.grid(row=4, column=0)
        two = Button(self, text="2")
        two.grid(row=4, column=1)
        thr = Button(self, text="3")
        thr.grid(row=4, column=2)
        mns = Button(self, text="-")
        mns.grid(row=4, column=3)

        zer = Button(self, text="0")
        zer.grid(row=5, column=0)
        dot = Button(self, text=".")
        dot.grid(row=5, column=1)
        equ = Button(self, text="=")
        equ.grid(row=5, column=2)
        pls = Button(self, text="+")
        pls.grid(row=5, column=3)

        self.pack()


def main():
    root = Tk()
```

```
    app = Example()

    root.mainloop()



if __name__ == '__main__':

    main()
```

Менеджер **grid()** используется для организации кнопок в контейнере рамки.

```
Style().configure("TButton", padding=(0, 5, 0, 5),

    font='serif 10')
```

Мы настроили **виджет кнопки** так, чтобы отображался специфический шрифт и применялся отступ (padding) в 3 пикселя.

```
self.columnconfigure(0, pad=3)

...

self.rowconfigure(0, pad=3)
```

Мы использовали методы **columnconfigure**() и **rowconfigure**() чтобы создать определенное пространство в сетке строк и столбцов. Благодаря этому шагу мы разделяем кнопки определенным пустым пространством.

```
entry = Entry(self)

entry.grid(row=0, columnspan=4, sticky=W+E)
```

Виджет графы ввода – это место, где будут отображаться цифры. Данный виджет расположен в первом ряду и охватывает все четыре столбца. Виджеты могут не занимать все пространство, которое выделяется клетками в созданной сетке.

Параметр **sticky** расширяет виджет в указанном направлении. В нашем случае, мы можем убедиться, что наш виджет графы ввода был расширен слева направо **W+E** (восток-запад).

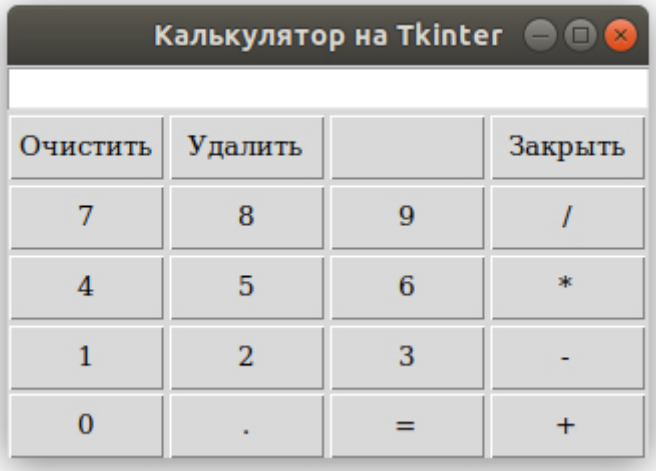
```
cls = Button(self, text="Очистить")

cls.grid(row=1, column=0)
```

Кнопка очистки установлена во второй строке и первом столбце. Стоит отметить, что строки и столбцы начинаются с нуля.

```
self.pack()
```

Метод pack() показывает виджет рамки и дает ей первоначальный размер. Если дополнительные параметры не указываются, размер будет таким, чтобы все дочерние виджеты могли поместиться. Этот метод компонует виджет рамки в верхнем корневом окне, которое также является контейнером. Менеджер **grid()** используется для организации кнопок в виджете рамки.



Семинарское занятие по теме 8

Тема: Создание графических приложений и компьютерных игр в Tk

Цель: научиться создавать интерактивные графические приложения в библиотеке Tk.

Задания (вопросы) для подготовки:

Создайте игру Змейка.

Этапы разработки игры на Tkinter

Размер каждого соединения змейки – 10 пикселей. Змейка управляется стрелками на клавиатуре. Изначально у змейки есть всего три соединения. Игра начинается мгновенно. Когда игра заканчивается, на экране появляется надпись «Игра Закончена».

Для начала, мы создаем Canvas виджет. Объектами в игре являются изображения. Мы используем методы **canvas** для создания объектов изображения. Также, мы используем методы canvas для обнаружения объектов на холсте при помощи тегов, и для обнаружения столкновений с другими объектами.

```
import sys

import random

from PIL import Image, ImageTk

from tkinter import Tk, Frame, Canvas, ALL, NW
```

```
class Cons:

    BOARD_WIDTH = 300
    BOARD_HEIGHT = 300
    DELAY = 100
    DOT_SIZE = 10
    MAX_RAND_POS = 27



class Board(Canvas):

    def __init__(self):
        super().__init__(
            width=Cons.BOARD_WIDTH, height=Cons.BOARD_HEIGHT,
            background="blue", highlightthickness=0
        )

        self.initGame()
        self.pack()

    def initGame(self):
        """
        Инициализация игры.
        """
```

```
        self.inGame = True
        self.dots = 3
        self.score = 0

        # Переменные для передвижения замеи.
        self.moveX = Cons.DOT_SIZE
        self.moveY = 0

        # Изначальные стартовые координаты яблоки.
        self.appleX = 100
        self.appleY = 190

        self.loadImages()

        self.createObjects()
        self.locateApple()
        self.bind_all("<Key>", self.onKeyPressed)
        self.after(Cons.DELAY, self.onTimer)

    def loadImages(self):
        """
        Подгружаем нужные изображения для игры.
        """

        try:
            self.idot = Image.open("dot.png")
```

```
            self.dot = ImageTk.PhotoImage(self.idot)
            self.ihead = Image.open("head.png")
            self.head = ImageTk.PhotoImage(self.ihead)
            self.iapple = Image.open("apple.png")
            self.apple = ImageTk.PhotoImage(self.iapple)

        except IOError as e:
            print(e)
            sys.exit(1)

    def createObjects(self):
        """
        Создание объектов на холсте.
        """

        self.create_text(
            30, 10, text="Счет: {0}".format(self.score),
            tag="score", fill="white"
        )

        self.create_image(
            self.appleX, self.appleY, image=self.apple,
            anchor=NW, tag="apple"
        )

        self.create_image(50, 50, image=self.head, anchor=NW, tag="head")
```

```
    self.create_image(30, 50, image=self.dot, anchor=NW, tag="dot")
    self.create_image(40, 50, image=self.dot, anchor=NW, tag="dot")

def checkAppleCollision(self):
    """
    Проверяем, не столкнулась ли голова змеи с яблоком.
    """

    apple = self.find_withtag("apple")
    head = self.find_withtag("head")

    x1, y1, x2, y2 = self.bbox(head)
    overlap = self.find_overlapping(x1, y1, x2, y2)

    for ovr in overlap:

        if apple[0] == ovr:

            self.score += 1
            x, y = self.coords(apple)
            self.create_image(x, y, image=self.dot, anchor=NW, tag="dot")
            self.locateApple()

def moveSnake(self):
    """
    Меняем положение змеи на холсте.
```

```
    """

    dots = self.find_withtag("dot")
    head = self.find_withtag("head")

    items = dots + head

    z = 0
    while z < len(items)-1:

        c1 = self.coords(items[z])
        c2 = self.coords(items[z+1])
        self.move(items[z], c2[0]-c1[0], c2[1]-c1[1])
        z += 1

    self.move(head, self.moveX, self.moveY)

def checkCollisions(self):
    """
    Проверка на столкновение змеи с другими объектами.
    """

    dots = self.find_withtag("dot")
    head = self.find_withtag("head")

    x1, y1, x2, y2 = self.bbox(head)
```

```
    overlap = self.find_overlapping(x1, y1, x2, y2)

    for dot in dots:
        for over in overlap:
            if over == dot:
                self.inGame = False

    if x1 < 0:
        self.inGame = False

    if x1 > Cons.BOARD_WIDTH - Cons.DOT_SIZE:
        self.inGame = False

    if y1 < 0:
        self.inGame = False

    if y1 > Cons.BOARD_HEIGHT - Cons.DOT_SIZE:
        self.inGame = False

def locateApple(self):
    """
    Распределяем яблоки по холсту (canvas).
    """

    apple = self.find_withtag("apple")
    self.delete(apple[0])
```

```
    r = random.randint(0, Cons.MAX_RAND_POS)
    self.appleX = r * Cons.DOT_SIZE
    r = random.randint(0, Cons.MAX_RAND_POS)
    self.appleY = r * Cons.DOT_SIZE

    self.create_image(
        self.appleX, self.appleY, anchor=NW,
        image=self.apple, tag="apple"
    )

def onKeyPressed(self, e):
    """
    Управляем змеей через стрелки клавиатуры.
    """

    key = e.keysym

    LEFT_CURSOR_KEY = "Left"
    if key == LEFT_CURSOR_KEY and self.moveX <= 0:
        self.moveX = -Cons.DOT_SIZE
        self.moveY = 0

    RIGHT_CURSOR_KEY = "Right"
    if key == RIGHT_CURSOR_KEY and self.moveX >= 0:
        self.moveX = Cons.DOT_SIZE
```

```
            self.moveY = 0

        RIGHT_CURSOR_KEY = "Up"
        if key == RIGHT_CURSOR_KEY and self.moveY <= 0:
            self.moveX = 0
            self.moveY = -Cons.DOT_SIZE

        DOWN_CURSOR_KEY = "Down"
        if key == DOWN_CURSOR_KEY and self.moveY >= 0:
            self.moveX = 0
            self.moveY = Cons.DOT_SIZE

    def onTimer(self):
        """
        Создает игровой цикл для каждого события таймера
        """

        self.drawScore()
        self.checkCollisions()

        if self.inGame:
            self.checkAppleCollision()
            self.moveSnake()
            self.after(Cons.DELAY, self.onTimer)
        else:
            self.gameOver()
```

```
    def drawScore(self):
        """
        Рисуем счет игры
        """

        score = self.find_withtag("score")
        self.itemconfigure(score, text="Счет: {0}".format(self.score))

    def gameOver(self):
        """
        Удаляем все объекты и выводим сообщение об окончании игры.
        """

        self.delete(ALL)
        self.create_text(self.winfo_width() / 2, self.winfo_height()/2,
            text="Игра закончилась со счетом {0}".format(self.score), fill="white")


class Snake(Frame):

    def __init__(self):
        super().__init__()
        self.master.title('Змейка')
        self.board = Board()
        self.pack()
```

```
def main():
    root = Tk()
    nib = Snake()
    root.mainloop()


if __name__ == '__main__':
    main()
```

В первую очередь, мы расшифруем некоторые переменные, использующиеся в нашей игре:

- BOARD_WIDTH и BOARD_HEIGHT обозначают размер игрового поля.
- DELAY – скорость игры;
- DOT_SIZE — размер яблока и соединения змейки;
- MAX_RAND_POS – создание случайной точки для яблока.

Метод **iniGame**() инициализирует переменные, загружает изображения игры и запускает функцию таймера.

```
self.createObjects()
self.locateApple()
```

Метод **createObject**() создает объекты на холсте, а **locateApple**() устанавливает яблоко на случайную точку на холсте.

```
self.bind_all("<Key>", self.onKeyPressed)
```

Мы назначаем клавиши клавиатуры при помощи метода **onKeyPressed**(). Для управления замеей в игре используются стрелки на клавиатуре.

```
try:
    self.idot = Image.open("dot.png")
    self.dot = ImageTk.PhotoImage(self.idot)
    self.ihead = Image.open("head.png")
```

```
    self.head = ImageTk.PhotoImage(self.ihead)

    self.iapple = Image.open("apple.png")

    self.apple = ImageTk.PhotoImage(self.iapple)


except IOError as e:

    print(e)

    sys.exit(1)
```

В этих строчках показана загрузка наших изображений. Это три изображения для нашей игры:

- голова (head.png);
- соединение змеи (dot.png);
- яблоко (apple.png).

```
def createObjects(self):

    """

    Создание объектов на холсте.

    """


    self.create_text(

        30, 10, text="Счет: {0}".format(self.score),

        tag="score", fill="white"

    )


    self.create_image(

        self.appleX, self.appleY, image=self.apple,

        anchor=NW, tag="apple"

    )

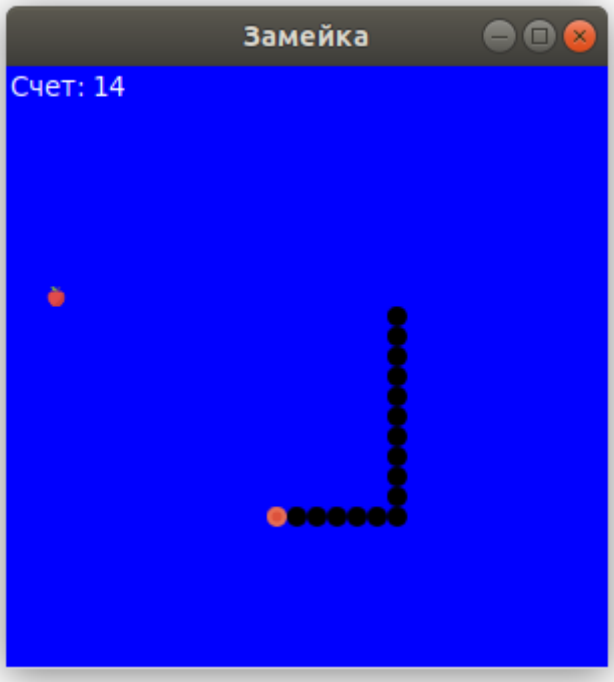

    self.create_image(50, 50, image=self.head, anchor=NW, tag="head")
```

```
self.create_image(30, 50, image=self.dot, anchor=NW, tag="dot")

self.create_image(40, 50, image=self.dot, anchor=NW, tag="dot")
```

С помощью метода **createObject**() мы создаем игровые объекты на холсте. Это canvas объекты. У них есть стартовые x и y координаты.

Параметр *image* позволяет изображениям появиться на холсте. Параметр anchor установлен на *NW* (Север и Запад), что означает ориентир на верхнюю левую точку холста. И этот шаг очень важен, если мы хотим отображать изображения около границ корневого окна tkinter.



Попробуйте удалить *anchor* и посмотрите, что произойдет. Параметр *tag* используется для идентификации объектов на холсте.

Один тег может использоваться для нескольких объектов на холсте.

Метод checkAppleCollision() позволяет нам узнать, съела ли змея яблоко или нет. Если да, мы добавляем одно соединение змейке и выполняем метод locateApple() для создания нового яблока на нашей игровой карте.

```
apple = self.find_withtag("apple")

head = self.find_withtag("head")
```

Метод **find_withtag**() позволяет нам найти объект на холсте, используя имя данного тега.

Нам нужно два объекта: голова змеи и яблоко.

Стоит отметить, что даже если присутствует только один объект с указанным тегом, метод вернет кортеж. Это для случая с объектом яблока. Позже объект яблока станет доступен следующим образом: *apple[0]*.

```
x1, y1, x2, y2 = self.bbox(head)

overlap = self.find_overlapping(x1, y1, x2, y2)
```

Метод **bbox**() возвращает точки границ объекта. С помощью метода **find_overlapping**() мы найдем столкновение объектов с этими координатами.

```
for ovr in overlap:



   if apple[0] == ovr:

      x, y = self.coords(apple)

      self.create_image(x, y, image=self.dot, anchor=NW, tag="dot")

      self.locateApple()
```

Если яблоко сталкивается с головой змеи, то мы создаем новое соединение для змейки на координатах яблока. Также, мы выполняем метод locateApple(), который удаляет старое яблоко из холста и создает новое в случайной точке на игровой карте.

Метод moveSnake() устанавливаем механизм управлениями змеей через клавиатуру. Чтобы понять, нужно посмотреть на то, как двигается змейка. Мы управляем ее головой. Мы можем изменить ее направление при помощи **клавиш стрелок на клавиатуре**. Остальные соединения двигаются за головой как **одна единая цепочка**.

```
z = 0

while z < len(items)-1:

   c1 = self.coords(items[z])

   c2 = self.coords(items[z+1])

   self.move(items[z], c2[0]-c1[0], c2[1]-c1[1])

   z += 1
```

Этот код позволяет соединениям (туловище змеи) двигаться одной цепью.

```
self.move(head, self.moveX, self.moveY)
```

Этот код меняет направление движения головы змеи. Значение атрибутов класса self.moveX и self.moveY меняются после каждого нажатия по стрелкам клавиатуры для изменения траектории движения змеи.

С помощью метода **checkColission**() мы определяем, когда змея удариться головой об свой хвост либо об стену.

```
x1, y1, x2, y2 = self.bbox(head)

overlap = self.find_overlapping(x1, y1, x2, y2)



for dot in dots:

    for over in overlap:

        if over == dot:

          self.inGame = False
```

Игра заканчивается, если голова змеи ударяется об свой хвост.

```
if y1 > HEIGHT - DOT_SIZE:

    self.inGame = False
```

Игра заканчивается, если змея ударяется о край игрового поля.

Метод **locateApple**() устанавливает новое яблоко в случайной точке на холсте и удаляет старое яблоко с игровой карты.

```
apple = self.find_withtag("apple")

self.delete(apple[0])
```

В этой части кода мы находим и удаляем яблоко, которое было съедено змейкой.

```
r = random.randint(0, Cons.MAX_RAND_POS)
```

Мы получаем случайное число от 0 до MAX_RAND_POS — 1

```
self.appleX = r * Cons.DOT_SIZE

...

self.appleY = r * Cons.DOT_SIZE
```

Здесь мы устанавливаем x и y координаты объекта яблока.

В методе **onKeyPressed**(), мы реагируем на какие клавиши нажал игрок.

```
LEFT_CURSOR_KEY = "Left"
```

```
if key == LEFT_CURSOR_KEY and self.moveX <= 0:

    self.moveX = -Cons.DOT_SIZE

    self.moveY = 0
```

Если мы нажали на стрелку влево, то обновляем значения из атрибутов self.moveX и self.moveY в соответствии с положением змеи на карте. Эти атрибуты используется в методе moveSnake(), который изменяет координаты объекта змеи. Стоит отметить, что если змея поворачивается направо, мы не можем сразу же развернуться влево.

```
def onTimer(self):

    """

    Создает игровой цикл для каждого события таймера

    """



    self.drawScore()

    self.checkCollisions()



    if self.inGame:

        self.checkAppleCollision()

        self.moveSnake()

        self.after(Cons.DELAY, self.onTimer)

    else:

        self.gameOver()
```

После каждого DELAY запускается метод **onTimer**(). Если мы в игре, мы запускаем три метода, которые являются основой логики игры. В противном случае игра заканчивается.

Таймер основан на методе **after**(), который запускает метод после *DELAY* только однажды. Чтобы повторно запускать таймер, мы рекурсивно запускаем метод **onTimer**().

```
def drawScore(self):

    """

    Рисуем счет игры

    """
```

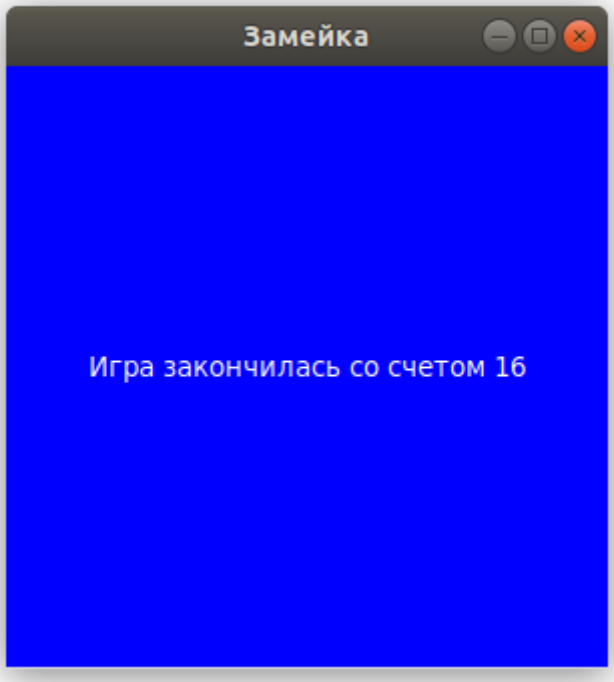
```
    score = self.find_withtag("score")
    self.itemconfigure(score, text="Счет: {0}".format(self.score))
```

Метод drawScore рисует счет игры на нашей игровой карте в верхнем левом углу.

```
def gameOver(self):
    """
    Удаляем все объекты и выводим сообщение об окончании игры.
    """

    self.delete(ALL)
    self.create_text(self.winfo_width() / 2, self.winfo_height()/2,
        text="Игра закончилась со счетом {0}".format(self.score), fill="white")
```

Если игра закончена, мы удаляем все объекты на холсте. Затем мы выводим по центру экрана финальный счет игрока.



Семинарское занятие по теме 9
Тема: Реализация web-приложения при использовании CGI-сценариев

Цель: изучить возможность реализации web-приложений при использовании CGI-сценариев на Python.

Задания (вопросы) для подготовки:

Модуль cgi

```
#!/usr/bin/python
# -*- coding: cp1251 -*-
import cgi, os

# анализ запроса
f = cgi.FieldStorage()
if f.has_key("a"):
  a = f["a"].value
else:
  a = "0"

# обработка запроса
b = str(int(a)+1)
mytext = open(os.environ["SCRIPT_FILENAME"]).read()
mytext_html = cgi.escape(mytext)

# формирование ответа
print """Content-Type: text/html

<html><head><title>Решение примера: %(b)s = %(a)s + 1</title></head>
<body>
%(b)s
<table width="80%%"><tr><td>
<form action="me.cgi" method="GET">
<input type="text" name="a" value="0" size="6">
<input type="submit" name="b" value="Обработать">
</form></td></tr></table>
<pre>
%(mytext_html)s
</pre>
</body></html>""" % vars()
```

Передача файлов на сервер (upload).

```
#!/usr/bin/env python
import cgi

form = cgi.FieldStorage()
file_contents = ""
if form.has_key("filename"):
  fileitem = form["filename"]
  if fileitem.file:
    file_contents = """<P>Содержимое переданного файла:
    <PRE>%s</PRE>""" % fileitem.file.read()

print """Content-Type: text/html

<HTML><HEAD><TITLE>Загрузка файла</TITLE></HEAD>
```

```
<BODY><H1>Загрузка файла</H1>
<P><FORM ENCTYPE="multipart/form-data"
ACTION="getfile.cgi" METHOD="POST">
<br>Файл: <INPUT TYPE="file" NAME="filename">
<br><INPUT TYPE="submit" NAME="button" VALUE="Передать файл">
</FORM>
%s
</BODY></HTML>""" % file_contents
```

Семинарское занятие по теме 10

Тема: Среды разработки web-приложений для Python

Цель: изучить возможности использования Python в web-приложениях.

Задания (вопросы) для подготовки:

1. Встроенный Web-сервер (ZServer).

2. Среда разработчика zope.

3. Поддержка языков сценариев Python, Perl и собственного DTML.

4. Модули для web-сервера mod_python.

Семинарское занятие по теме 1

Тема: Сетевые приложения на Python

Цель: изучить возможности Python для реализации клиент-серверного приложения.

Задания (вопросы) для подготовки:

Проектирование клиент-серверного приложения

Сервер:

```
import socket, string

def do_something(x):
  lst = map(None, x);
  lst.reverse();
  return string.join(lst, "")

HOST = ""                 # localhost
PORT = 33333
srv = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
srv.bind((HOST, PORT))
while 1:
  print "Слушаю порт 33333"
  srv.listen(1)
  sock, addr = srv.accept()
  while 1:
    pal = sock.recv(1024)
    if not pal:
      break
    print "Получено от %s:%s:" % addr, pal
    lap = do_something(pal)
    print "Отправлено %s:%s:" % addr, lap
```

```
    sock.send(lap)
  sock.close()
```

Клиент:

```
import socket

HOST = ""                 # удаленный компьютер (localhost)
PORT = 33333              # порт на удаленном компьютере
sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
sock.connect((HOST, PORT))
sock.send("ПАЛИНДРОМ")
result = sock.recv(1024)
sock.close()
print "Получено:", result
```

Модуль smtplib

Сообщения электронной почты в Интернете передаются от клиента к серверу и между серверами в основном по протоколу **SMTP** (**S** imple **M** ail **T** ransfer **P** rotocol, простой протокол передачи почты). Протокол SMTP и ESMTP (расширенный вариант SMTP) описаны в RFC 821 и RFC 1869. Для работы с SMTP в стандартной библиотеке модулей имеется модуль smtplib. Для того чтобы начать SMTP-соединение с сервером электронной почты, необходимо в начале создать объект для управления SMTP-сессией с помощью конструктора класса SMTP:

```
smtplib.SMTP([host[, port]])
```

Параметры host и port задают адрес и порт SMTP-сервера, через который будет отправляться почта. По умолчанию, port=25. Если host задан, конструктор сам установит соединение, иначе придется отдельно вызывать метод connect(). Экземпляры класса SMTP имеют методы для всех распространенных команд SMTP-протокола, но для отправки почты достаточно вызова конструктора и методов sendmail() и quit():

```
# -*- coding: cp1251 -*-
from smtplib import SMTP
fromaddr = "student@mail.ru"      # От кого
toaddr = "rnd@onego.ru"           # Кому
message = """From: Student <%(fromaddr)s>
To: Lecturer <%(toaddr)s>
Subject: From Python course student
MIME-Version: 1.0
Content-Type: text/plain; charset=Windows-1251
Content-Transfer-Encoding: 8bit

Здравствуйте! Я изучаю курс по языку Python и
отправляю письмо его автору.
"""
```

```
connect = SMTP('mail.onego.ru')
connect.set_debuglevel(1)
connect.sendmail(fromaddr, toaddr, message % vars())
connect.quit()
```

Модуль poplib

Еще один протокол - **POP3** (**P** ost **O** ffice **P** rotocol, почтовый протокол) - служит для приема почты из почтового ящика на сервере (протокол определен в RFC 1725). Для работы с почтовым сервером требуется установить с ним соединение и, подобно рассмотренному выше примеру, с помощью SMTP-команд получить требуемые сообщения. Объект-соединение POP3 можно установить посредством конструктора класса POP3 из модуля poplib:

```
poplib.POP3(host[, port])
```

Где host - адрес POP3-сервера, port - порт на сервере (по умолчанию 110), pop_obj - объект для управления сеансом работы с POP3-сервером.

Основные методы для работы с POP3-соединением:

```
import poplib, email
# Учетные данные пользователя:
SERVER = "pop.server.com"
USERNAME = "user"
USERPASSWORD = "secretword"

p = poplib.POP3(SERVER)
print p.getwelcome()
# этап идентификации
print p.user(USERNAME)
print p.pass_(USERPASSWORD)
# этап транзакций
response, lst, octets = p.list()
print response
for msgnum, msgsize in [i.split() for i in lst]:
  print "Сообщение %(msgnum)s имеет длину %(msgsize)s" % vars()
  print "UIDL =", p.uidl(int(msgnum)).split()[2]
  if int(msgsize) > 32000:
    (resp, lines, octets) = p.top(msgnum, 0)
  else:
    (resp, lines, octets) = p.retr(msgnum)
  msgtxt = "\n".join(lines)+"\n\n"
  msg = email.message_from_string(msgtxt)
  print "* От: %(from)s\n* Кому: %(to)s\n* Тема: %(subject)s\n" % msg
  # msg содержит заголовки сообщения или все сообщение (если оно небольшое)

# этап обновления
print p.quit()
```

При выполнении сценарий выведет на экран примерно следующее.

```
+OK POP3 pop.server.com server ready
+OK User name accepted, password please
+OK Mailbox open, 68 messages
+OK Mailbox scan listing follows
Сообщение 1 имеет длину 4202
UIDL = 4152a47e00000004
* От: online@kaspersky.com
* Кому: user@server.com
* Тема: KL Online Activation

...

+OK Sayonara
```

Семинарское занятие по теме 2
Тема: Модули для клиента WWW
Цель: изучение стандартных средств Python для доступа к объектам WWW.
Задания (вопросы) для подготовки:

Функции для загрузки сетевых объектов

Простой случай получения WWW-объекта по известному URL:

```
import urllib
doc = urllib.urlopen("http://python.onego.ru").read()
print doc[:40]
```

Функция urllib.urlopen() создает файлоподобный объект, который читает методом read(). Другие методы этого объекта: readline(), readlines(), fileno(), close() работают как и у обычного файла, а также есть метод info(), который возвращает соответствующий полученному с сервера Message-объект. Этот объект можно использовать для получения дополнительной информации:

```
>>> import urllib
>>> f = urllib.urlopen("http://python.onego.ru")
>>> print f.info()
Date: Sat, 25 Dec 2004 19:46:11 GMT
Server: Apache/1.3.29 (Unix) PHP/4.3.10
Content-Type: text/html; charset=windows-1251
Content-Length: 4291
>>> print f.info()['Content-Type']
text/html; charset=windows-1251
```

С помощью функции urllib.urlopen() можно делать и более сложные вещи, например, передавать web-серверу данные формы. Как известно, данные заполненной web-формы могут быть переданы на web-сервер с использованием метода GET или метода POST. Метод GET связан с кодированием всех передаваемых параметров после знака "?" в URL,

а при методе POST данные передаются в теле HTTP-запроса. Оба варианта передачи представлены ниже:

```
import urllib
data = {"search": "Python"}
enc_data = urllib.urlencode(data)

# метод GET
f = urllib.urlopen("http://searchengine.com/search" + "?" + enc_data)
print f.read()

# метод POST
f = urllib.urlopen("http://searchengine.com/search", enc_data)
print f.read()
```

В некоторых случаях данные имеют повторяющиеся имена. В этом случае в качестве параметра urllib.urlencode() можно использовать вместо словаря последовательность пар имя-значение:

```
>>> import urllib
>>> data = [("n", "1"), ("n", "3"), ("n", "4"), ("button", "Привет"),]
>>> enc_data = urllib.urlencode(data)
>>> print enc_data
n=1&n=3&n=4&button=%F0%D2%C9%D7%C5%D4
```

Модуль urllib позволяет загружать web-объекты через прокси-сервер. Если ничего не указывать, будет использоваться прокси-сервер, который был задан принятым в конкретной ОС способом. В Unix прокси-серверы задаются в переменных окружения http_proxy, ftp_proxy и т.п., в Windows прокси-серверы записаны в реестре, а в Mac OS они берутся из конфигурации Internet. Задать прокси-сервер можно и как именованный параметр proxies к urllib.urlopen():

```
# Использовать указанный прокси
proxies = {'http': 'http://www.proxy.com:3128'}
f = urllib.urlopen(some_url, proxies=proxies)
# Не использовать прокси
f = urllib.urlopen(some_url, proxies={})
# Использовать прокси по умолчанию
f = urllib.urlopen(some_url, proxies=None)
f = urllib.urlopen(some_url)
```

Функция urlretrieve() позволяет записать заданный URL сетевой объект в файл. Она имеет следующие параметры:

```
urllib.urlretrieve(url[, filename[, reporthook[, data]]])
```

Здесь url - URL сетевого объекта, filename - имя локального файла для помещения объекта, reporthook - функция, которая будет вызываться для сообщения о состоянии загрузки, data - данные для метода POST (если он используется). Функция возвращает кортеж (filepath, headers) , где filepath - имя локального файла, в который закачан объект, headers - результат метода info() для объекта, возвращенного urlopen().

Для обеспечения интерактивности функция urllib.urlretrieve() вызывает время от времени функцию, заданную в reporthook(). Этой функции передаются три аргумента: количество принятых блоков, размер блока и общий размер принимаемого объекта в байтах (если он неизвестен, этот параметр равен -1).

```
FILE = 'boost-1.31.0-9.src.rpm'
URL = 'http://download.fedora.redhat.com/pub/fedora/linux/core/3/SRPMS/' + FILE

def download(url, file):
  import urllib, time
  start_t = time.time()

  def progress(bl, blsize, size):
    dldsize = min(bl*blsize, size)
    if size != -1:
      p = float(dldsize) / size
      try:
        elapsed = time.time() - start_t
        est_t = elapsed / p - elapsed
      except:
        est_t = 0
      print "%6.2f %% %6.0f s %6.0f s %6i / %-6i bytes" % (
          p*100, elapsed, est_t, dldsize, size)
    else:
      print "%6i / %-6i bytes" % (dldsize, size)

  urllib.urlretrieve(URL, FILE, progress)

download(URL, FILE)
```

Эта программа выведет примерно следующее (процент от полного объема закачки, прошедшие секунды, предполагаемое оставшееся время, закачанные байты, полное количество байтов):

```
0.00 %      1 s      0 s      0 / 6952309 bytes
0.12 %      5 s   3941 s   8192 / 6952309 bytes
0.24 %      7 s   3132 s  16384 / 6952309 bytes
0.35 %     10 s   2864 s  24576 / 6952309 bytes
0.47 %     12 s   2631 s  32768 / 6952309 bytes
```

```
0.59 %     15 s   2570 s  40960 / 6952309 bytes
0.71 %     18 s   2526 s  49152 / 6952309 bytes
0.82 %     20 s   2441 s  57344 / 6952309 bytes
```

Возможности urllib2

Одна из полезных возможностей этих модулей - доступ к web-объектам, требующий авторизации.

Создание собственного открывателя URL с помощью модуля urllib2:

```
import urllib2

# Подготовка идентификационных данных
authinfo = urllib2.HTTPBasicAuthHandler()
authinfo.add_password('My page', 'localhost', 'user1', 'secret')

# Доступ через прокси
proxy_support = urllib2.ProxyHandler({'http' : 'http://localhost:8080'})

# Создание нового открывателя с указанными обработчиками
opener = urllib2.build_opener(proxy_support,
                              authinfo,
                              urllib2.CacheFTPHandler)
# Установка поля с названием клиента
opener.addheaders = [('User-agent', 'Mozilla/5.0')]

# Установка нового открывателя по умолчанию
urllib2.install_opener(opener)

# Использование открывателя
f = urllib2.urlopen('http://localhost/mywebdir/')
print f.read()[:100]
```

XML-RPC сервер

Программа на Python, реализующую сервер, с помощью протокола XML-RPC. В языке Python вызов удаленной функции по синтаксису ничем не отличается от вызова обычной функции:

```
import xmlrpclib

# Установить соединение
req = xmlrpclib.ServerProxy("http://localhost:8000")

try:
    # Вызвать удаленную функцию
    print req.add(1, 3)
except xmlrpclib.Error, v:
    print "ERROR",
```

XML-RPC-сервер (для того чтобы попробовать пример выше, необходимо сначала запустить сервер):

```
from SimpleXMLRPCServer import SimpleXMLRPCServer
srv = SimpleXMLRPCServer(("localhost", 8000))  # Запустить сервер
srv.register_function(pow)  # Зарегистрировать функцию
srv.register_function(lambda x,y: x+y, 'add')  # И еще одну
srv.serve_forever()  # Обслуживать запросы
```

Семинарское занятие по теме 3

Тема: Многопоточное программирование

Цель: изучить возможности многопоточного программирования, используя объекты класса Thread.

Задания (вопросы) для подготовки:

Пример многопоточной программы

В следующем примере создается два дополнительных потока, которые выводят на стандартный вывод каждый свое:

```
import threading

            def proc(n):
              print "Процесс", n

            p1 = threading.Thread(target=proc, name="t1", args=["1"])
            p2 = threading.Thread(target=proc, name="t2", args=["2"])
            p1.start()
            p2.start()
```

Функции модуля threading

В модуле threading, который здесь используется, есть функции, позволяющие получить информацию о потоках:

- activeCount() Возвращает количество активных в настоящий момент экземпляров класса Thread. Фактически, это len(threading.enumerate()).
- currentThread() Возвращает текущий объект-поток, то есть соответствующий потоку управления, который вызвал эту функцию. Если поток не был создан через модуль threading, будет возвращен объект-поток с сокращенной функциональностью (dummy thread object).
- enumerate() Возвращает список активных потоков. Завершившиеся и еще не начатые потоки не входят в список.

Жизнью потоков можно управлять вызовом методов:

- start() Дает потоку жизнь.
- run() Этот метод представляет действия, которые должны быть выполнены в потоке.
- join([timeout]) Поток, который вызывает этот метод, приостанавливается, ожидая завершения потока, чей метод вызван. Параметр timeout (число с плавающей точкой) позволяет указать время ожидания (в секундах), по истечении которого приостановленный поток продолжает свою работу независимо от завершения потока, чей метод join был вызван. Вызывать join() некоторого потока можно много раз. Поток не может вызвать метод join() самого себя. Также нельзя ожидать завершения еще не запущенного потока. Слово "join" в переводе с английского означает "присоединить", то есть, поток, вызвавший join(), желает, чтобы поток по завершении присоединился к вызывающему метод потоку.
- getName() Возвращает имя потока. Для главного потока это "MainThread".
- setName(name) Присваивает потоку имя name.
- isAlive() Возвращает истину, если поток работает (метод run() уже вызван, но еще не завершился).
- isDaemon() Возвращает истину, если поток имеет признак демона. Программа на Python завершается по завершении всех потоков, не являющихся демонами. Главный поток демоном не является.
- setDaemon(daemonic) Устанавливает признак daemonic того, что поток является демоном. Начальное значение этого признака заимствуется у потока, запустившего данный. Признак можно изменять только для потоков, которые еще не запущены.

Таймер

Класс threading.Timer представляет действие, которое должно быть выполнено через заданное время. Этот класс является подклассом класса threading.Thread, поэтому запускается также методом start(). Следующий простой пример, печатающий на стандартном выводе Hello, world! поясняет сказанное:

```
def hello():
              print "Hello, world!"

            t = Timer(30.0, hello)
            t.start()
```

Тупиковая ситуация (deadlock)

Замки применяются для управления доступом к ресурсу, который нельзя использовать совместно. В программе таких ресурсов может быть несколько. При работе с замками важно хорошо продумать, не зайдет ли выполнение программы в **тупик** (deadlock) из-за того, что двум потокам потребуются одни и те же ресурсы, но ни тот, ни другой не смогут их получить, так как они уже получили замки. Такая ситуация проиллюстрирована в следующем примере:

```
import threading, time

                resource = {'A': threading.Lock(), 'B': threading.Lock()}

                def proc(n, rs):
                 for r in rs:
                   print "Процесс %s запрашивает ресурс %s" % (n, r)
                   resource[r].acquire()
                   print "Процесс %s получил ресурс %s" % (n, r)
                   time.sleep(1)
                 print "Процесс %s выполняется" % n
                 for r in rs:
                   resource[r].release()
                 print "Процесс %s закончил выполнение" % n

                p1 = threading.Thread(target=proc, name="t1", args=["1",
"AB"])
                p2 = threading.Thread(target=proc, name="t2", args=["2",
"BA"])
                p1.start()
                p2.start()
                p1.join()
                p2.join()
```

Семинарское занятие по теме 4

Тема: Управление потоками

Цель: практическое применение возможностей управления нитями и потоками в Python.

Задания (вопросы) для подготовки:

Методы Объекта класса Lock:

- acquire([blocking=True]) Делает запрос на запирание замка. Если параметр blocking не указан или является истиной, то поток будет ожидать освобождения замка. Если параметр не был задан, метод не возвратит значения. Если blocking был задан и истинен, метод возвратит True (после успешного овладения замком). Если блокировка не требуется (то есть задан blocking=False), метод вернет True, если замок не был заперт и им успешно овладел данный поток. В противном случае будет возвращено False.
- release() Запрос на отпирание замка.
- locked() Возвращает текущее состояние замка (True - заперт, False - открыт). Следует иметь в виду, что даже если состояние замка только что проверено, это не означает, что он сохранит это состояние до следующей команды.

Тупиковая ситуация:

```
import threading, time

                resource = {'A': threading.Lock(), 'B': threading.Lock()}

                def proc(n, rs):
                 for r in rs:
                   print "Процесс %s запрашивает ресурс %s" % (n, r)
                   resource[r].acquire()
                   print "Процесс %s получил ресурс %s" % (n, r)
                   time.sleep(1)
                 print "Процесс %s выполняется" % n
                 for r in rs:
                   resource[r].release()
                 print "Процесс %s закончил выполнение" % n

                p1 = threading.Thread(target=proc, name="t1", args=["1",
"AB"])
                p2 = threading.Thread(target=proc, name="t2", args=["2",
"BA"])
                p1.start()
                p2.start()
                p1.join()
                p2.join()
```

Семафоры

Конструктор класса threading.Semaphore принимает в качестве (необязательного) аргумента начальное состояние счетчика (по умолчанию оно равно 1, что соответствует замку класса Lock). Методы acquire() и release() действуют аналогично описанным выше одноименным методам у замков. Семафор может применяться для охраны ограниченного ресурса. Например, с его помощью можно вести **пул** соединений с базой данных. Пример такого использования семафора:

```
from threading import BoundedSemaphore
            maxconnections = 5
            # Подготовка семафора
            pool_sema = BoundedSemaphore(value=maxconnections)

            # Внутри потока:

            pool_sema.acquire()
            conn = connectdb()
            # ... использование соединения ...
            conn.close()
            pool_sema.release()
```

методы экземпляров класса threading.Event:

- set() Устанавливает внутренний флаг, сигнализирующий о наступлении события. Все ждущие данного события потоки выходят из состояния ожидания.
- clear() Сбрасывает флаг. Все события, которые вызывают метод wait() этого объекта-события, будут находиться в состоянии ожидания до тех пор, пока флаг сброшен, или по истечении заданного таймаута.
- isSet() Возвращает состояние флага.
- wait([timeout]) Переводит поток в состояние ожидания, если флаг сброшен, и сразу возвращается, если флаг установлен. Аргумент timeout задает таймаут в секундах, по истечении которого ожидание прекращается, даже если событие не наступило.

Условия

Условия используются для оповещения потоков о прибытии новой порции данных (организуется связь производитель - потребитель, producer - consumer):

```
import threading

            cv = threading.Condition()

            class Item:
              """Класс-контейнер для элементов, которые будут потребляться
              в потоках"""
              def __init__(self):
                self._items = []
              def is_available(self):
                return len(self._items) > 0
              def get(self):
                return self._items.pop()
              def make(self, i):
                self._items.append(i)

            item = Item()

            def consume():
              """Потребление очередного элемента (с ожиданием его
появления)"""
              cv.acquire()
              while not item.is_available():
                cv.wait()
              it = item.get()
              cv.release()
              return it

            def consumer():
              while True:
                print consume()

            def produce(i):
              """Занесение нового элемента в контейнер и оповещение
потоков"""
              cv.acquire()
              item.make(i)
              cv.notify()
              cv.release()
```

```
p1 = threading.Thread(target=consumer, name="t1")
p1.setDaemon(True)
p2 = threading.Thread(target=consumer, name="t2")
p2.setDaemon(True)
p1.start()
p2.start()
produce("ITEM1")
produce("ITEM2")
produce("ITEM3")
produce("ITEM4")
p1.join()
p2.join()
```

События

Еще одним способом коммуникации между объектами являются **события**. Экземпляры класса threading.Event могут быть использованы для передачи информации о наступлении некоторого события от одного потока одному или нескольким другим потокам. Объекты-события имеют внутренний флаг, который может находиться в установленном или сброшенном состоянии. При своем создании флаг события находится в сброшенном состоянии. Если флаг в установленном состоянии, ожидания не происходит: поток, вызвавший метод wait() для ожидания события, просто продолжает свою работу. Ниже приведены методы экземпляров класса threading.Event:

- set() Устанавливает внутренний флаг, сигнализирующий о наступлении события. Все ждущие данного события потоки выходят из состояния ожидания.
- clear() Сбрасывает флаг. Все события, которые вызывают метод wait() этого объекта-события, будут находиться в состоянии ожидания до тех пор, пока флаг сброшен, или по истечении заданного таймаута.
- isSet() Возвращает состояние флага.
- wait([timeout]) Переводит поток в состояние ожидания, если флаг сброшен, и сразу возвращается, если флаг установлен. Аргумент timeout задает таймаут в секундах, по истечении которого ожидание прекращается, даже если событие не наступило.

В следующем примере условия используются для оповещения потоков о прибытии новой порции данных (организуется связь производитель - потребитель, producer - consumer):

```
import threading

            cv = threading.Condition()

            class Item:
              """Класс-контейнер для элементов, которые будут потребляться
              в потоках"""
              def __init__(self):
                self._items = []
              def is_available(self):
```

```
            return len(self._items) > 0
          def get(self):
            return self._items.pop()
          def make(self, i):
            self._items.append(i)

        item = Item()

        def consume():
          """Потребление очередного элемента (с ожиданием его
появления)"""
          cv.acquire()
          while not item.is_available():
            cv.wait()
          it = item.get()
          cv.release()
          return it

        def consumer():
          while True:
            print consume()

        def produce(i):
          """Занесение нового элемента в контейнер и оповещение
потоков"""
          cv.acquire()
          item.make(i)
          cv.notify()
          cv.release()

        p1 = threading.Thread(target=consumer, name="t1")
        p1.setDaemon(True)
        p2 = threading.Thread(target=consumer, name="t2")
        p2.setDaemon(True)
        p1.start()
        p2.start()
        produce("ITEM1")
        produce("ITEM2")
        produce("ITEM3")
        produce("ITEM4")
        p1.join()
        p2.join()
```

В этом примере условие cv отражает наличие необработанных элементов в контейнере item. Функция produce() "производит" элементы, а consume(), работающая внутри потоков, "потребляет". Стоит отметить, что в приведенном виде программа никогда не закончится, так как имеет бесконечный цикл в потоках, а в главном потоке - ожидание завершения этих потоков. Еще одна особенность - признак демона, установленный с помощью метода setDaemon() объекта-потока до его старта.

Очередь

Процесс, показанный в предыдущем примере, имеет значение, достойное отдельного модуля. Такой модуль в стандартной библиотеке языка Python есть, и он называется Queue. Помимо исключений - Queue.Full (очередь переполнена) и Queue.Empty (очередь пуста) - модуль определяет класс Queue, заведующий собственно очередью.

```
import threading, Queue

            item = Queue.Queue()

            def consume():
              """Потребление очередного элемента (с ожиданием его
появления)"""
              return item.get()

            def consumer():
              while True:
                print consume()

            def produce(i):
              """Занесение нового элемента в контейнер и оповещение
потоков"""
              item.put(i)

            p1 = threading.Thread(target=consumer, name="t1")
            p1.setDaemon(True)
            p2 = threading.Thread(target=consumer, name="t2")
            p2.setDaemon(True)
            p1.start()
            p2.start()
            produce("ITEM1")
            produce("ITEM2")
            produce("ITEM3")
            produce("ITEM4")
            p1.join()
            p2.join()
```

Модуль thread

По сравнению с модулем threading, модуль thread предоставляет низкоуровневый доступ к потокам. Многие функции модуля threading, который рассматривался до этого, реализованы на базе модуля thread. Здесь стоит сделать некоторые замечания по применению потоков вообще. Использование потоков имеет особенности:

- Исключение KeyboardInterrupt (прерывание от клавиатуры) может быть получено любым из потоков, если в поставке Python нет модуля signal (для обработки сигналов).
- Не все встроенные функции, блокированные ожиданием ввода, позволяют другим потокам работать. Правда, основные функции вроде time.sleep(), select.select(), метод read() файловых объектов не блокируют другие потоки.
- Невозможно прервать метод acquire(), так как исключение KeyboardInterrupt возбуждается только после возврата из этого метода.
- Нежелательно, чтобы главный поток завершался раньше других потоков, так как не будут выполнены необходимые деструкторы и даже части finally в операторах try-

finally. Это связано с тем, что почти все операционные системы завершают приложение, у которого завершился главный поток.

Визуализация работы потоков

Иллюстрация параллельности выполнения потоков, используя возможности библиотеки графических примитивов Tkinter. Несколько потоков наперегонки увеличивают размеры прямоугольника некоторого цвета. Цветом победившего потока окрашивается кнопка Go:

```
import threading, time, sys
            from Tkinter import Tk, Canvas, Button, LEFT, RIGHT, NORMAL,
DISABLED

            global champion

            # Задается дистанция, цвет полосок и другие параметры
            distance = 300
            colors =
["Red","Orange","Yellow","Green","Blue","DarkBlue","Violet"]
            nrunners = len(colors)      # количество дополнительных потоков
            positions = [0] * nrunners  # список текущих позиций
            h, h2 = 20, 10              # параметры высоты полосок

            def run(n):
              """Программа бега n-го участника (потока)"""
              global champion
              while 1:
                for i in range(10000):           # интенсивные вычисления
                  pass
                graph_lock.acquire()
                positions[n] += 1                # передвижение на шаг
                if positions[n] == distance:     # если уже финиш
                  if champion is None:           # и чемпион еще не
определен,
                    champion = colors[n]         # назначается чемпион
                  graph_lock.release()
                  break
                graph_lock.release()

            def ready_steady_go():
              """Инициализация начальных позиций и запуск потоков"""
              graph_lock.acquire()
              for i in range(nrunners):
                positions[i] = 0
                threading.Thread(target=run, args=[i,]).start()
              graph_lock.release()

            def update_positions():
              """Обновление позиций"""
              graph_lock.acquire()
              for n in range(nrunners):
                c.coords(rects[n], 0, n*h, positions[n], n*h+h2)
              tk.update_idletasks()  # прорисовка изменений
              graph_lock.release()

            def quit():
              """Выход из программы"""
```

```
            tk.quit()
            sys.exit(0)

        # Прорисовка окна, основы для прямоугольников и самих
прямоугольников,
        # кнопок для пуска и выхода
        tk = Tk()
        tk.title("Соревнование потоков")
        c = Canvas(tk, width=distance, height=nrunners*h, bg="White")
        c.pack()
        rects = [c.create_rectangle(0, i*h, 0, i*h+h2, fill=colors[i])
                 for i in range(nrunners)]
        go_b = Button(text="Go", command=tk.quit)
        go_b.pack(side=LEFT)
        quit_b = Button(text="Quit", command=quit)
        quit_b.pack(side=RIGHT)

        # Замок, регулирующий доступ к функции пакета Tk
        graph_lock = threading.Lock()

        # Цикл проведения соревнований
        while 1:
          go_b.config(state=NORMAL), quit_b.config(state=NORMAL)
          tk.mainloop()             # Ожидание нажатия клавиш
          champion = None
          ready_steady_go()
          go_b.config(state=DISABLED), quit_b.config(state=DISABLED)
          # Главный поток ждет финиша всех участников
          while sum(positions) < distance*nrunners:
            update_positions()
          update_positions()
          go_b.config(bg=champion)     # Кнопка окрашивается в цвет
победителя
          tk.update_idletasks()
```

Семинарское занятие по теме 5
Тема: Модуль DB API 2.0
Цель: изучить возможности DB API 2.0.
Задания (вопросы) для подготовки:

Объект-соединение

Объект-соединение, получаемый в результате успешного вызова функции connect(), должен иметь следующие методы:

- close() Закрывает соединение с базой данных.
- commit() Завершает транзакцию.
- rollback() Откатывает начатую транзакцию (восстанавливает исходное состояние). Закрытие соединения при незавершенной транзакции автоматически производит откат транзакции.
- cursor() Возвращает объект-курсор, использующий данное соединение. Если база данных не поддерживает курсоры, модуль сопряжения должен их имитировать.

Объект-курсор

Курсор (от англ. cursor - CURrrent Set Of Records, текущий набор записей) служит для работы с результатом запроса. Результатом запроса обычно является одна или несколько прямоугольных таблиц со столбцами-полями и строками-записями. Приложение может читать и обрабатывать полученные таблицы и записи в таблице по одной, поэтому в курсоре хранится информация о текущей таблице и записи. Конкретный курсор в любой момент времени связан с выполнением одной SQL-инструкции.

Атрибуты объекта-курсора тоже определены DB-API:

- arraysize Атрибут, равный количеству записей, возвращаемых методом fetchmany(). По умолчанию равен 1.
- callproc(procname[, params]) Вызывает хранимую процедуру procname с параметрами из изменчивой последовательности params. Хранимая процедура может изменить значения некоторых параметров последовательности. Метод может возвратить результат, доступ к которому осуществляется через fetch - методы.
- close() Закрывает объект-курсор.
- description Этот доступный только для чтения атрибут является последовательностью из семиэлементных последовательностей. Каждая из этих последовательностей содержит информацию, описывающую один столбец результата:
- ```
  (name, type_code, display_size, internal_size, precision, scale,
  null_ok)
  ```

  Первые два элемента (имя и тип) обязательны, а вместо остальных (размер для вывода, внутренний размер, точность, масштаб, возможность задания пустого значения) может быть значение None. Этот атрибут может быть равным None для операций, не возвращающих значения.

- execute(operation[, parameters]) Исполняет запрос к базе данных или команду СУБД. Параметры (parameters) могут быть представлены в принятой в базе данных нотации в соответствии с атрибутом paramstyle, описанным выше.
- executemany(operation, seq_of_parameters) Выполняет серию запросов или команд, подставляя параметры в заданный шаблон. Параметр seq_of_parameters задает последовательность наборов параметров.
- fetchall() Возвращает все (или все оставшиеся) записи результата запроса.
- fetchmany([size]) Возвращает следующие несколько записей из результатов запроса в виде последовательности последовательностей. Пустая последовательность означает отсутствие данных. Необязательный параметр size указывает количество возвращаемых записей (реально возвращаемых записей может быть меньше). По умолчанию size равен атрибуту arraysize объекта-курсора.
- fetchone() Возвращает следующую запись (в виде последовательности) из результата запроса или None при отсутствии данных.
- nextset() Переводит курсор к началу следующего набора данных, полученного в результате запроса (при этом часть записей в предыдущем наборе может остаться


  ```

непрочитанной). Если наборов больше нет, возвращает None. Не все базы данных поддерживают возврат нескольких наборов результатов за одну операцию.

- rowcount Количество записей, полученных или затронутых в результате выполнения последнего запроса. В случае отсутствия execute-запросов или невозможности указать количество записей равен -1.
- setinputsizes(sizes) Предопределяет области памяти для параметров, используемых в операциях. Аргумент sizes задает последовательность, где каждый элемент соответствует одному входному параметру. Элемент может быть объектом-типом соответствующего параметра или целым числом, задающим длину строки. Он также может иметь значение None, если о размере входного параметра ничего нельзя сказать заранее или он предполагается очень большим. Метод должен быть вызван до execute-методов.
- setoutputsize(size[, column]) Устанавливает размер буфера для выходного параметра из столбца с номером column. Если column не задан, метод устанавливает размер для всех больших выходных параметров. Может использоваться, например, для получения **больших бинарных объектов** (**B** inary **L** arge **O** bject, **BLOB**).

Объекты-типы

DB-API 2.0 предусматривает названия для объектов-типов, используемых для описания полей базы данных:

Объект	Тип
STRING	Строка и символ
BINARY	Бинарный объект
NUMBER	Число
DATETIME	Дата и время
ROWID	Идентификатор записи
None	NULL-значение (отсутствующее значение)

Семинарское занятие по теме 6
Тема: Работа с базой данных из Python-приложения
Цель: изучить возможности модуля sqlite для работы с базами данных в Python.
Задания (вопросы) для подготовки:

Создание базы данных

Для создания базы данных нужно установить, какие таблицы (и другие объекты, например индексы) в ней будут храниться, а также определить структуры таблиц (имена и типы полей). Задача - создание базы данных, в которой будет храниться телепрограмма. В этой базе будет таблица со следующими полями:

- tvdate,
- tvweekday,
- tvchannel,
- tvtime1,
- tvtime2,
- prname,
- prgenre.

Здесь tvdate - дата, tvchannel - канал, tvtime1 и tvtime2 - время начала и конца передачи, prname - название, prgenre - жанр. Конечно, в этой таблице есть функциональная зависимость (tvweekday вычисляется на основе tvdate и tvtime1), но из практических соображений БД к нормальным формам приводиться не будет. Кроме того, будет создана таблица с названиями дней недели (устанавливает соответствие между номером дня и днем недели):

- weekday,
- wdname.

Следующий сценарий создаст таблицу в базе данных (в случае с SQLite заботиться о создании базы данных не нужно: файл создастся автоматически. Для других баз данных необходимо перед этим создать базу данных, например, SQL-инструкцией CREATE DATABASE):

```
import sqlite as db

   c = db.connect(database="tvprogram")
   cu = c.cursor()

   try:
     cu.execute("""
       CREATE TABLE tv (
         tvdate DATE,
         tvweekday INTEGER,
         tvchannel VARCHAR(30),
         tvtime1 TIME,
         tvtime2 TIME,
         prname VARCHAR(150),
         prgenre VARCHAR(40)
       );
   """)
   except db.DatabaseError, x:
     print "Ошибка: ", x
   c.commit()

   try:
     cu.execute("""
       CREATE TABLE wd (
         weekday INTEGER,
         wdname VARCHAR(11)
```

```
      );
  """)
  except db.DatabaseError, x:
    print "Ошибка: ", x
  c.commit()
  c.close()
```

Наполнение базы данных

```
weekdays = ["Воскресенье", "Понедельник", "Вторник", "Среда",
               "Четверг", "Пятница", "Суббота", "Воскресенье"]

  import sqlite as db

  c = db.connect(database="tvprogram")
  cu = c.cursor()
  cu.execute("""DELETE FROM wd;""")
  cu.executemany("""INSERT INTO wd VALUES (%s, %s);""",
            enumerate(weekdays))
  c.commit()
  c.close()
```

Выборки из базы данных

Базы данных создаются для удобства хранения и извлечения больших объемов. Следующий нехитрый пример позволяет проверить, правильно ли были введены в таблицу дни недели:

```
import sqlite as db

  c = db.connect(database="tvprogram")
  cu = c.cursor()
  cu.execute("SELECT weekday, wdname FROM wd ORDER BY weekday;")
  for i, n in cu.fetchall():
    print i, n
```

Если все было сделано правильно, получится:

```
0 Воскресенье
  1 Понедельник
  2 Вторник
  3 Среда
  4 Четверг
  5 Пятница
  6 Суббота
  7 Воскресенье
```

Выборка телепрограммы:

```
import sqlite as db

   c = db.connect(database="tvprogram")
   cu = c.cursor()
   cu.execute("""
   SELECT tvdate, tvtime1, wd.wdname, tvchannel, prname, prgenre
   FROM tv, wd
   WHERE wd.weekday = tvweekday
   ORDER BY tvdate, tvtime1;""")
   for rec in cu.fetchall():
     dt = rec[0] + rec[1]
     weekday = rec[2]
     channel = rec[3]
     name = rec[4]
     genre = rec[5]
     print "%s, %02i.%02i.%04i %s %02i:%02i %s (%s)" % (
            weekday, dt.day, dt.month, dt.year, channel,
            dt.hour, dt.minute, name, genre)
```

Семинарское занятие по теме 7

Тема: Работа с файлами и сообщениями

Цель: изучить возможности Python для работы с современными форматами данных с использованием стандартных библиотек.

Задания (вопросы) для подготовки:

Формат CSV

Чтение CSV-файла и записи другого, где числа второго столбца увеличены на единицу:

```
import csv
input_file = open("pr.csv", "rb")
rdr = csv.reader(input_file)
output_file = open("pr1.csv", "wb")
wrtr = csv.writer(output_file)
for rec in rdr:
  try:
    rec[1] = int(rec[1]) + 1
  except:
    pass
  wrtr.writerow(rec)
input_file.close()
output_file.close()
```

В результате получится файл pr1.csv следующего содержания:

```
name,number,text
a,2,something here
b,3,"one, two, three"
c,4,"no commas here"
```

Определение двух классов для более удобного чтения и записи значений с использованием словаря. Вызовы конструкторов следующие:

class DictReader(csvfile, fieldnames[, restkey=None[, restval=None[, dialect='excel']]])

Пакет email

Модули пакета email помогут разобрать, изменить и сгенерировать сообщение в формате RFC 2822. Наиболее часто RFC 2822 применяется в сообщениях электронной почты в Интернете. В пакете есть несколько модулей:

1. Message
2. Parser
3. Header
4. Generator
5. Utils

Разбор сообщения. Класс Message

Класс Message - центральный во всем пакете email. Он определяет методы для работы с сообщением, которое состоит из заголовка (header) и тела (payload). Поле заголовка имеет название и значение, разделенное двоеточием (двоеточие не входит ни в название, ни в значение). Названия полей нечувствительны к регистру букв при поиске значения, хотя хранятся с учетом регистра. В классе также определены методы для доступа к некоторым часто используемым сведениям (кодировке сообщения, типу содержимого и т.п.). Следует заметить, что сообщение может иметь одну или несколько частей, в том числе вложенных друг в друга. Например, сообщение об ошибке доставки письма может содержать исходное письмо в качестве вложения.

Пример наиболее употребительных методов экземпляров класса Message:

```
>>> import email
>>> input_file = open("pr1.eml")
>>> msg = email.message_from_file(input_file)
```

Здесь используется функция email.message_from_file() для чтения сообщения из файла pr1.eml. Сообщение можно получить и из строки с помощью функции email.message_from_string(). А теперь следует произвести некоторые операции над этим сообщением (не стоит обращать внимания на странные имена - сообщение было взято из папки СПАМ). Доступ к полям по имени осуществляется так:

```
>>> print msg['from']
```

```
"felton olive" <zinakinch@thecanadianteacher.com>
>>> msg.get_all('received')
['from mail.onego.ru\n\tby localhost with POP3 (fetchmail-6.2.5
polling mail.onego.ru account spam)\n\tfor spam@localhost
(single-drop); Wed, 01 Sep 2004 15:46:33 +0400 (MSD)',
'from thecanadianteacher.com ([222.65.104.100])\n\tby mail.onego.ru
(8.12.11/8.12.11) with SMTP id i817UtUN026093;\n\tWed, 1 Sep 2004
11:30:58 +0400']
```

Некоторые важные данные можно получить в готовом виде, например, тип содержимого, кодировку:

```
>>> msg.get_content_type()
'text/plain'
>>> print msg.get_main_type(), msg.get_subtype()
text plain
>>> print msg.get_charset()
None
>>> print msg.get_params()
[('text/plain', ''), ('charset', 'us-ascii')]
>>> msg.is_multipart()
False
```

или список полей:

```
>>> print msg.keys()
['Received', 'Received', 'Message-ID', 'Date', 'From', 'User-Agent',
'MIME-Version', 'To', 'Subject', 'Content-Type',
'Content-Transfer-Encoding', 'Spam', 'X-Spam']
```

Так как сообщение состоит из одной части, можно получить его тело в виде строки:

```
>>> print msg.get_payload()
sorgeloosheid  hullw ifesh nozama decompresssequenceframes

Believe it or not, I have tried several sites to b"_"uy presription
medication. I should say that currently you are still be the best amony
...
```

Вирусные сообщения

Сообщение состоит из нескольких частей. Это сообщение порождено вирусом. Оно состоит из двух частей: HTML-текста и вложенного файла с расширением cpl. Для доступа к частям сообщения используется метод walk(), который обходит все его части. Попутно следует собрать типы содержимого (в списке parts), поля Content-Type (в ct_fields) и имена файлов (в filenames):

```
import email
parts = []
ct_fields = []
filenames = []
f = open("virus.eml")
msg = email.message_from_file(f)
for submsg in msg.walk():
  parts.append(submsg.get_content_type())
  ct_fields.append(submsg.get('Content-Type', ''))
  filenames.append(submsg.get_filename())
  if submsg.get_filename():
      print "Длина файла:", len(submsg.get_payload())
f.close()
print parts
print ct_fields
print filenames
```

В результате получилось:

```
Длина файла: 31173
['multipart/mixed', 'text/html', 'application/octet-stream']
['multipart/mixed;\n        boundary="--------hidejpxkblmvuwfplzue"',
'text/html; charset="us-ascii"',
'application/octet-stream; name="price.cpl"']
[None, None, 'price.cpl']
```

Из списка parts можно увидеть, что само сообщение имеет тип multipart/mixed, тогда как две его части - text/html и application/octet-stream соответственно. Только с последней частью связано имя файла (price.cpl). Файл читается методом get_payload() и вычисляется его длина.

Семинарское занятие по теме 8
Тема: Язык XML
Цель: изучить структуру документа XML.
Задания (вопросы) для подготовки:

Структура XML

```
<?xml version="1.0" encoding="iso-8859-1"?>
<expression>
  <operation type="+">
    <operand>2</operand>
    <operand>
      <operation type="*">
        <operand>3</operand>
        <operand>4</operand>
      </operation>
    </operand>
  </operation>
</expression>
```

Формирование XML-документа

Концептуально существуют два пути обработки XML-документа: последовательная обработка и работа с объектной моделью документа. В первом случае обычно используется **SAX** (Simple API for XML, простой программный интерфейс для XML). Работа SAX заключается в чтении источников данных (input source) XML-анализаторами (XML-reader) и генерации последовательности событий (events), которые обрабатываются объектами-обработчиками (handlers). SAX дает последовательный доступ к XML-документу. Во втором случае анализатор XML строит **DOM** (Document Object Model, объектная модель документа), предлагая для XML-документа конкретную объектную модель. В рамках этой модели узлы DOM-дерева доступны для произвольного доступа,а для переходов между узлами предусмотрен ряд методов.

С помощью SAX документ сформируется так:

```
import sys
from xml.sax.saxutils import XMLGenerator
g = XMLGenerator(sys.stdout)
g.startDocument()
g.startElement("expression", {})
g.startElement("operation", {"type": "+"})
g.startElement("operand", {})
g.characters("2")
g.endElement("operand")
g.startElement("operand", {})
g.startElement("operation", {"type": "*"})
g.startElement("operand", {})
g.characters("3")
g.endElement("operand")
g.startElement("operand", {})
g.characters("4")
g.endElement("operand")
g.endElement("operation")
g.endElement("operand")
g.endElement("operation")
g.endElement("expression")
g.endDocument()
```

Построение дерева объектной модели документа может выглядеть, так:

```
from xml.dom import minidom
dom = minidom.Document()
e1 = dom.createElement("expression")
dom.appendChild(e1)
p1 = dom.createElement("operation")
p1.setAttribute('type', '+')
x1 = dom.createElement("operand")
x1.appendChild(dom.createTextNode("2"))
p1.appendChild(x1)
e1.appendChild(p1)
p2 = dom.createElement("operation")
p2.setAttribute('type', '*')
```

```
x2 = dom.createElement("operand")
x2.appendChild(dom.createTextNode("3"))
p2.appendChild(x2)
x3 = dom.createElement("operand")
x3.appendChild(dom.createTextNode("4"))
p2.appendChild(x3)
x4 = dom.createElement("operand")
x4.appendChild(p2)
p1.appendChild(x4)
print dom.toprettyxml()
```

Анализ XML-документа

Для работы с готовым XML-документом нужно воспользоваться XML-анализаторами. Анализ XML-документа с порождением объекта класса Document происходит всего в одной строчке, с помощью функции parse(). Здесь стоит заметить, что кроме стандартного пакета xml можно поставить пакет PyXML или альтернативные коммерческие пакеты:

```
import xml.dom.minidom
dom = xml.dom.minidom.parse("expression.xml")

dom.normalize()

def output_tree(node, level=0):
  if node.nodeType == node.TEXT_NODE:
    if node.nodeValue.strip():
      print ". "*level, node.nodeValue.strip()
  else:   # ELEMENT_NODE или DOCUMENT_NODE
    atts = node.attributes or {}
    att_string = ", ".join(
        ["%s=%s " % (k, v)  for k, v in atts.items()])
    print ". "*level, node.nodeName, att_string
    for child in node.childNodes:
      output_tree(child, level+1)

output_tree(dom)
```

В этом примере дерево выводится с помощью определенной функции output_tree(), которая принимает на входе узел и вызывается рекурсивно для всех вложенных узлов.

В результате получается следующее:

```
 #document
. expression
. . operation type=+
. . . operand
. . . . 2
. . . operand
. . . . operation type=*
. . . . . operand
```

. 3
. . . . . operand
. 4

Пространства имен

Они позволяют составлять XML-документы из кусков различных схем. Nаким образом в XML-документ можно включить кусок HTML, указав во всех элементах HTML принадлежность особому пространству имен.

Следующий пример XML-кода показывает синтаксис пространств имен (файл foaf.rdf):

```
<?xml version="1.0" encoding="UTF-8"?>
<rdf:RDF
   xmlns:dc="http://http://purl.org/dc/elements/1.1/"
   xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
   xmlns:foaf="http://xmlns.com/foaf/0.1/"
   xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
>
  <rdf:Description rdf:nodeID="_:jCBxPziO1">
    <foaf:nick>donna</foaf:nick>
    <foaf:name>Donna Fales</foaf:name>
    <rdf:type rdf:resource="http://xmlns.com/foaf/0.1/Person"/>
  </rdf:Description>
</rdf:RDF>
```

6.2. Задания и упражнения для самостоятельной работы

Цель самостоятельной работы студентов заключается в глубоком, полном усвоении учебного материала и в развитии навыков самообразования. Самостоятельная работа студента включает: работу с текстами, основной литературой, учебно-методическим пособием, нормативными материалами, дополнительной литературой, в том числе материалами Интернета, а также проработка конспектов лекций, написание докладов, рефератов, участие в работе семинаров, студенческих научных конференциях.

Задания для самостоятельной работы:

№	Наименование модуля (темы) дисциплины	Вопросы для самостоятельной работы
Модуль 1. Объектно-ориентированное программирование		
1	Основные понятия и принципы объектно-ориентированного программирования	Вопросы: 1.Декомпозиция в Python. 2.Классы объектов встроенных типов в Python. 3.Вызовы конструкторов классов и методов-деструкторов классов в Python. 4.Инкапсуляция. Доступ к свойствам и атрибутам классов в Python. 5.Сокрытие днных атрибута класса в Python.
2	Отношения между классами	

		6.Полиморфизм в Python. 7.Имитация типов в Python. 8.Наследование в Python. 9.Множественное наследование от нескольких классов в Python. 10.Порядок разрешения методов (method resolution order) в Python. 11.Контейнеры в Python. 12 Итераторы в Python. 13.Ассоциации и циклические ссылки в Python. 14.Слабые ссылки в Python. Модуль weakref. 15.Статические методы для применения описателей (descriptors) в Python. 16.Методы класса в модуле tree пакета nltk. 17.Метаклассы в Python. Динамическое создание классов в функции-фабрике классов. 18.Создание собственных мультиметодов с помощью модуля Multimethod. Задачи: 1. Создайте класс People, конструктор* которого принимает аргумент name — имя человека. Класс должен содержать метод get_name, который возвращает имя заглавными буквами. * Метод __init__(), который мы изучили в первом уроке про классы, называется конструктором. 2. Создайте объект people класса People, а затем выведите на экран имя человека заглавными буквами с помощью метода get_name.
Модуль 2. Функциональное программирование		
3	Модули и встроенные функции	Вопросы: 1. Сервисы периода выполнения. Модули: sys, atexit, copy, traceback, math, cmath, random, time, calendar, datetime, sets, array, struct, itertools, locale, gettext.
4	Модули стандартной библиотеки	
5	Функциональная программа	

6	Итераторы и простые генераторы	2. Поддержка цикла разработки. Модули: pdb, hotshot, profile, unittest, pydoc. Пакеты docutils, distutils. 3. Взаимодействие с ОС (файлы, процессы). Модули: os, os.path, getopt, glob, popen2, shutil, select, signal, stat, tempfile. 4. Обработка текстов. Модули: string, re, StringIO, codecs, difflib, mmap, sgmllib, htmllib, htmlentitydefs. Пакет xml. 5. Многопоточные вычисления. Модули: threading, thread, Queue. 6. Хранение данных. Архивация. Модули: pickle, shelve, anydbm, gdbm, gzip, zlib, zipfile, bz2, csv, tarfile. 7. Платформо-зависимые модули. Для UNIX: commands, pwd, grp, fcntl, resource, termios, readline, rlcompleter. Для Windows: msvcrt, _winreg, winsound. 8. Поддержка сети. Протоколы Интернет. Модули: cgi, Cookie, urllib, urlparse, httplib, smtplib, poplib, telnetlib, socket, asyncore. Примеры серверов: SocketServer, BaseHTTPServer, xmlrpclib, asynchat. 9. Поддержка Internet. Форматы данных. Модули: quopri, uu, base64, binhex, binascii, rfc822, mimetools, MimeWriter, multifile, mailbox. Пакет email. 10. Графический интерфейс. Модуль Tkinter. Задачи: 1. Создайте функцию map, которая добавит к каждой комнате в списке rooms элемент с именем square содержащий её площадь. rooms = [{"name": "кухня", "length": 6, "width": 4}, {"name": "комната 1", "length": 5.5, "width": 4.5}, {"name": "комната 2", "length": 5, "width": 4}, {"name": "комната 3", "length": 7, "width": 6.3},] rooms = map()

		2. Используйте map и reduce, чтобы посчитать площадь квартиры, состоящей из комнат rooms. from functools import reduce rooms = [{"name": "кухня", "length": 6, "width": 4}, {"name": "комната 1", "length": 5.5, "width": 4.5}, {"name": "комната 2", "length": 5, "width": 4}, {"name": "комната 3", "length": 7, "width": 6.3},] rooms = map() square = reduce() Вопросы: 1.Программы в функциональном стиле 2.Композиция функций. 3.Функция: определение и вызов. 4.Список формальных параметров и тело определения функции. 5.Функция одного аргумента. 6.Функция двух аргументов. 7.Функция с одним обязательным аргументом, с одним, имеющим значение по умолчанию и неопределенным числом именованных аргументов. 8.Функция произвольного числа аргументов. 9.Функция с обычными (позиционными) и именованными аргументами. 10.Обработка последовательностей. Списковые включения.
Модуль 3. Создание графического интерфейса пользователя		
7	Создание графического интерфейса в библиотеке Tk	Создайте калькулятор для расчета индекса массы тела.

8	Создание графических приложений и компьютерных игр в Tk	
Модуль 4. Web-программирование		
9	CGI-сценарии	1. CGI-сценарии.
10	Среды разработки web для Python	2. Zope и его объектная модель. 3. Создания сервера web-приложений Zope. 4. Документы Zope. 5. Модули для web-сервера mod_python. 6. Методы передачи данных из заполненной в браузере формы Web-серверу. 7. Установленные Web-сервером переменные окружения.
Модуль 5. Программирование сетевых приложений		
11	Клиент-серверное приложение	1. Работа с сокетами.
12	Доступ к объектам WWW	2. SMTP-соединение с сервером электронной почты. 3. Прием почты из почтового ящика на сервере POP3. 4. Модули для клиента WWW. 5. Функции для загрузки сетевых объектов. 6. Функции для анализа URL. 7. Возможности модуля urllib2. 8. XML-RPC сервер. 9. Протокол XML-RPC.
Модуль 6. Многопоточное программирование		
13	Основы многопоточности в Python	Вопросы:
14	Передача данных между потоками	1. Виды многопоточности. 2. Запуск и завершение потоков. 3. Потоки управления. 4. Состояния потоков. 5. Приоритеты, блокировки. 6. Конструктор класса threading.Thread.

		7. Управление жизнью потоков. 8. Таймер. 9. Передача данных между потоками. 10. Управление доступом к потокам. 11. Замки. 12. Тупиковая ситуация (deadlock). 13. Семафоры. 14. События. 15. Условия. 16. Очередь. 17. Визуализация работы потоков. Задачи: 1. В директории лежат входные текстовые файлы, проименованные следующим образом: in_<N>.dat, где N - натуральное число. Каждый файл состоит из двух строк. В первой строке - число, обозначающее действие, а во второй - числа с плавающей точкой, разделенные пробелом. Действия могут быть следующими: 1 – сложение 2 – умножение 3 - сумма квадратов Необходимо написать многопоточное приложение, которое выполнит требуемые действия над числами и сумму результатов запишет в файл out.dat. Название рабочей директории передается в виде аргумента рабочей строки. В реализации приветствуется использование полиморфизма и паттернов проектирования. 2. Разработайте многопоточное приложение, выполняющее вычисление произведения матриц A (m×n) и B (n×k). Элементы cij матрицы произведения C = A×B вычисляются параллельно p однотипными потоками. Если некоторый поток уже вычисляет элемент cij матрицы C, то следующий приступающий к вычислению поток выбирает для расчета элемент cij+1, если j<k, и ci+1k, если j=k. Выполнив вычисление элемента

		матрицы-произведения, поток проверяет, нет ли элемента, который еще не рассчитывается. Если такой элемент есть, то приступает к его расчету. В противном случае отправляет (пользовательское) сообщение о завершении своей работы и приостанавливает своё выполнение. Главный поток, получив сообщения о завершении вычислений от всех потоков, выводит результат на экран и запускает поток, записывающий результат в конец файла-протокола. В каждом потоке должна быть задержка в выполнении вычислений (чтобы дать возможность поработать всем потокам). Синхронизацию потоков между собой организуйте через критическую секцию или мьютекс.
Модуль 7. Программирование запросов к базе данных из Python		
15	Интерфейсы модуля расширения для работы с базой данных	1. Описание DB API 2.0. 2. Интерфейс модуля.
16	Работа с базой данных из Python-приложения	3. Доступ к базе данных. 4. Объект-соединение. 5. Объект-курсор. 6. Объекты-типы. 7. Работа с базой данных из Python-приложения. 8. Модуль sqlite. 9. Создание базы данных. 10. Наполнение базы данных. 11. Выборки из базы данных. 12. Модули, поддерживающие DB-API 2.0 для MySQL, Oracle, Microsoft ActiveX Data Objects, Sybase.
Модуль 8. Расширенные возможности работы с файлами		
17	Работа с файлами и сообщениями	Вопросы: 1. Работа с файлами.
18	Работа с XML файлами	2. Работа с табличными данными. 3. Формат CSV. 4. Пакет email. 5. Разбор сообщения.

		6. Формирование сообщения. 7. Разбор поля заголовка. 8. XML файлы. 9. Формирование XML-документа. 10. Анализ XML-документа. 11. Пространства имен. Задачи: 1. Напишите программу man.py, которая будет создавать двоичный файл scientist.bin с краткой информацией о человеке. В файле должны содержаться следующие данные: 1) Имя – строка до 50 символов (байтов). 2) Фамилия – строка до 50 символов (байтов). 3) Год рождения – целое число типа short. 4) Год смерти – целое число типа short. Сделайте так, чтобы программа записала в файл информацию об Альберте Эйнштейне.

Контроль самостоятельной работы осуществляется на занятиях в ходе семинаров.

.

6.3. Перечень тем докладов, сообщений, презентаций и домашних заданий студентов

Учебным планом не предусмотрено.

6.4. Перечень тем (задания) для курсовой работы / Перечень тем (задания) для рейтинговой работы

Задания для выполнения рейтинговой работы 1

1. Разработка программного продукта для решения прикладных задач

Все темы (задания), требования к содержанию и оформлению рейтинговой / курсовой работы, критерии их оценки размещены в Методических рекомендациях по написанию рейтинговой / курсовой работы.

6.5. Иные материалы (по усмотрению преподавателя)

Вопросы для подготовки к промежуточной аттестации

1. Декомпозиция в Python.
2. Классы объектов встроенных типов в Python.
3. Вызовы конструкторов классов и методов-деструкторов классов в Python.
4. Инкапсуляция. Доступ к свойствам и атрибутам классов в Python.
5. Сокрытие данных атрибута класса в Python.

6. Полиморфизм в Python.
7. Имитация типов в Python.
8. Наследование в Python.
9. Множественное наследование от нескольких классов в Python.
10. Порядок разрешения методов (method resolution order) в Python.
11. Контейнеры в Python.
12. Итераторы в Python.
13. Ассоциации и циклические ссылки в Python.
14. Слабые ссылки в Python. Модуль weakref.
15. Статические методы для применения описателей (descriptors) в Python.
16. Методы класса в модуле tree пакета nltk.
17. Метаклассы в Python. Динамическое создание классов в функции-фабрике классов.
18. Создание собственных мультиметодов с помощью модуля Multimethod.
19. Функции преобразования типов и классы: coerce, str, repr, int, list, tuple, long, float, complex, dict, super, file, bool, object.
20. Числовые и строковые функции: abs, divmod, ord, pow, len, chr, unichr, hex, oct, cmp, round, unicode.
21. Функции обработки данных: apply, map, filter, reduce, zip, range, xrange, max, min, iter, enumerate, sum.
22. Функции определения свойств: hash, id, callable, issubclass, isinstance, type.
23. Функции для доступа к внутренним структурам: locals, globals, vars, intern, dir.
24. Функции компиляции и исполнения: eval, execfile, reload, __import__, compile.
25. Функции ввода-вывода: input, raw_input, open.
26. Функции для работы с атрибутами: getattr, setattr, delattr, hasattr.
27. Функции-"украшатели" методов классов: staticmethod, classmethod, property.
28. Прочие функции: buffer, slice.
29. Модули: sys, atexit, copy, traceback, math, cmath, random, time, calendar, datetime, sets, array, struct, itertools, locale, gettext.
30. Модули: pdb, hotshot, profile, unittest, pydoc. Пакеты docutils, distutils.
31. Модули: os, os.path, getopt, glob, popen2, shutil, select, signal, stat, tempfile.
32. Модули: pickle, shelve, anydbm, gdbm, gzip, zlib, zipfile, bz2, csv, tarfile.
33. Платформо-зависимые модули. Для UNIX: commands, pwd, grp, fcntl, resource, termios, readline, rlcompleter. Для Windows: msvcrt, _winreg, winsound.
34. Модули: cgi, Cookie, urllib, urlparse, httplib, smtplib, poplib, telnetlib, socket, asyncore. Примеры серверов: SocketServer, BaseHTTPServer, xmlrpclib, asynchat.
35. Модули: quopri, uu, base64, binhex, binascii, rfc822, mimetools, MimeWriter, multifile, mailbox. Пакет email.
36. Функция apply()
37. Функции range() и xrange()
38. Функция map()
39. Функция filter().
40. Функция sum().
41. Функция reduce().
42. Функция zip().
43. Функция iter().
44. Функция enumerate().
45. Функция sorted().

46. Функция itertools.chain().
47. Функция itertools.repeat().
48. Функция itertools.count().
49. Функция itertools.cycle().
50. Функции itertools.imap(), itertools.starmap() и itertools.ifilter().
51. Функции itertools.takewhile() и itertools.dropwhile().
52. Функция itertools.izip().
53. Функция itertools.groupby().
54. Функция itertools.tee().
55. Итератор, определенный пользователем.
56. Формирование функции-генератора.
57. Генераторное выражение.
58. Библиотека Xoltar toolkit с модулем functional.
59. Виджеты и их свойства.
60. Менеджер геометрии Pack.
61. Менеджер геометрии Place.
62. Менеджер геометрии Grid.
63. Text – многострочное текстовое поле.
64. Метод bind().
65. События.
66. Создание графических примитивов.
67. Canvas. Идентификаторы, теги и анимация.
68. CGI-сценарии.
69. Метод GET.
70. Метод POST.
71. Встроенный Web-сервер (ZServer).
72. Среда разработчика zope.
73. Поддержка языков сценариев Python, Perl и собственного DTML.
74. Модули для web-сервера mod_python.
75. Flask.
76. Django Framework.
77. Модуль socket.
78. Функция socket.getservbyname().
79. Модуль smtplib.
80. Конструктор класса POP3 из модуля poplib.
81. Модуль poplib.
82. Функция urllib.urlopen().
83. Функции urllib.urlencode().
84. Модуль urllib.
85. Функция urlretrieve().
86. Функция quote(s, safe='/').
87. Функциональность модуля urllib2.
88. Виды многопоточности.
89. Запуск и завершение потоков.
90. Состояния потоков.
91. Приоритеты.
92. Блокировки.
93. Функции модуля threading.

94. Конструктор класса threading.Thread.
95. Методы управления жизнью потоков.
96. Класс threading.Timer.
97. Передача данных между потоками.
98. Синхронизация.
99. Замки. Объект класса Lock.
100. Тупиковая ситуация (deadlock).
101. Семафоры. Конструктор класса threading.Semaphore.
102. События.
103. Экземпляры класса threading.Event.
104. Условия. Экземпляры класса threading.Condition.
105. Очередь. Модуль Queue.
106. Модуль thread.
107. Методы объекта-соединения.
108. Функция connect().
109. Атрибуты объекта-курсора.
110. Названия для объектов-типов, используемых для описания полей базы данных.
111. Реляционная база данных.
112. Подключение к базе данных.
113. Создание одного или нескольких курсоров (вызов метода объекта-соединения cursor() с получением объекта-курсора).
114. Исполнение команды или запроса (вызов метода execute() или его вариантов).
115. Получение результатов запроса (вызов метода fetchone() или его вариантов).
116. Завершение транзакции или ее откат (вызов метода объекта-соединения commit() или rollback()).
117. Закрытие подключения методом close().
118. Сценарии SQL для создания таблиц (CREATE TABLE).
119. Сценарии SQL для наполнения таблиц (INSERT).
120. Сценарии SQL для выборки данных (SELECT).
121. Менеджер контекста with.
122. Запись в файл с помощью print.
123. Чтение из текстового файла.
124. Двоичные файлы и тип bytes.
125. Модуль pickle.
126. Модуль struct.
127. Формат CSV.
128. Функция reader(csvfile[, dialect='excel'[, fmtparam]]).
129. Функция writer(csvfile[, dialect='excel'[, fmtparam]]).
130. Последовательная обработка и работа с объектной моделью документа.
131. Формирование XML-документа.
132. XML-анализаторы. Пакет PyXML.
133. Как создать поток в Python, используя стандартный модуль threading?
134. Как обеспечить синхронизацию потоков в Python, чтобы избежать проблем с доступом к общим данным?
135. Какие проблемы могут возникнуть при использовании глобальных переменных в многопоточных приложениях, и как их можно решить?

136. Какие методы блокировки предоставляет модуль threading, и в каких случаях их следует использовать?

137. Как реализовать передачу данных между потоками в Python? Приведите пример использования очереди

138. Какие существуют подходы к реализации параллельного выполнения кода в Python, помимо использования потоков?

139. Какие существуют подходы к обработке исключений в многопоточных программах на Python? Как избежать возможных проблем при обработке ошибок?

7. Оценочные средства для проведения текущего контроля и промежуточной аттестации обучающихся

7.1 Примерные оценочные средства, включая тестовые оценочные задания для проведения текущего контроля и промежуточной аттестации обучающихся по дисциплине (модулю) приведены в Приложении 1 к рабочей программе дисциплины.

7.2 Оценочные средства для проведения промежуточной аттестации обучающихся по дисциплине (модулю) включают следующие разделы:

- перечень компетенций, формируемых в процессе освоения учебной дисциплины;
- описание показателей и критериев оценивания компетенций, описание шкал оценивания;
- типовые контрольные задания или иные материалы, необходимые для оценки результатов обучения по учебной дисциплине, обеспечивающих достижение планируемых результатов освоения образовательной программы;
- методические материалы, определяющие процедуры оценивания результатов обучения по учебной дисциплине, обеспечивающих достижение планируемых результатов освоения образовательной программы.

8. Литература

8.1. Основная литература:

1. Сузи Р.А. Язык программирования Python: учебное пособие. - 2-е изд., испр. - М.: Интернет-Университет Информационных Технологий (ИНТУИТ), Бином. Лаборатория, 2007. - 327 с. – [Электронный ресурс] - https://biblioclub.ru/index.php?page=book&id=233288

2. Шелудько В.М. Язык программирования высокого уровня Python : функции, структуры данных, дополнительные модули: учебное пособие - Ростов-на-Дону, Таганрог: Южный федеральный университет, 2017. - 108 с. – [Электронный ресурс] - https://biblioclub.ru/index.php?page=book&id=500060

8.2. Дополнительная литература:

1. Карякин М.И., Ватульян К.А., Мнухин Р.М. Технологии программирования и компьютерный практикум на языке Python: учебное пособие - Ростов-на-Дону, Таганрог: Южный федеральный университет, 2022. - 244 с. – [Электронный ресурс] - https://biblioclub.ru/index.php?page=book&id=698687

2. Хахаев И.А. Практикум по алгоритмизации и программированию на Python : курс: учебное пособие. - 2-е изд., исправ. - М.: Национальный Открытый

Университет «ИНТУИТ», 2016. - 179 с. – [Электронный ресурс] - https://biblioclub.ru/index.php?page=book&id=429256

9. Перечень ресурсов информационно-телекоммуникационной сети «Интернет»

9.1. Официальный сайт Университета: адрес ресурса - http://www.muiv.ru/, на котором содержатся сведения об образовательной организации и ее подразделениях, локальные нормативные акты, сведения о реализуемых образовательных программах, их учебно-методическом и материально-техническом обеспечении, а также справочная, оперативная и иная информация. Через официальный сайт обеспечивается доступ всех участников образовательного процесса к различным сервисам и ссылкам, в том числе образовательному порталу «Электронный университет», ресурсам электронной библиотечной системы (далее - ЭБС), системе дистанционного обучения (далее – СДО) и др.;

9.2. Образовательный портал «Электронный университет»: адрес ресурса - https://e.muiv.ru/ на платформе «Moodle»;

9.3. Система дистанционного образования: адрес ресурса – https://lms.muiv.ru позволяет реализовать проведение всех видов занятий, процедур оценки результатов обучения, реализация которых предусмотрена с применением электронного обучения, дистанционных образовательных технологий.

9.4. www.elibrary.ru Научная электронная библиотека eLIBRARY.RU – крупнейший российский информационный портал в области науки, технологии, медицины и образования, содержащий рефераты и полные тексты

9.5. https://www.muiv.ru/ Московский университет имени С.Ю. Витте

9.6. https://github.com/ Крупнейший веб-сервис для хостинга IT-проектов и их совместной разработки

9.7. https://www.ixbt.com/ специализированный российский информационно-аналитический сайт с самыми актуальными новостями из сферы IT

10. Методические указания для обучающихся

10.1. Преподавание дисциплины осуществляется в соответствии с Федеральным государственным образовательным стандартом высшего образования, утвержденным Минобрнауки России, по направлению подготовки «Прикладная информатика».

Основными формами получения и закрепления знаний по данной дисциплине являются занятия лекционного и семинарского типа, самостоятельная работа обучающегося, в том числе под руководством преподавателя, прохождение рубежного контроля (модульного тестирования).

Учебный материал по дисциплине «Высокоуровневые методы программирования» разделен на восемь модулей:

Модуль 1. Объектно-ориентированное программирование;
Модуль 2. Функциональное программирование;
Модуль 3. Создание графического интерфейса пользователя;
Модуль 4. Web-программирование;

Модуль 5. Программирование сетевых приложений;
Модуль 6. Многопоточное программирование;
Модуль 7. Программирование запросов к базе данных из Python;
Модуль 8. Расширенные возможности работы с файлами.

Эти модули изучаются на всех формах обучения, реализуемых для данного направления подготовки.

Основной объем часов по изучению дисциплины согласно учебным планам приходится на самостоятельную работу обучающихся. Самостоятельная работа включает в себя изучение учебной, учебно-методической и специальной литературы, её конспектирование, подготовку к занятиям семинарского типа, текущему контролю и промежуточной аттестации (зачету или (и) экзамену).

Текущий контроль успеваемости по учебной дисциплине и промежуточная аттестация осуществляются в соответствие с Положением о текущем контроле успеваемости и промежуточной аттестации обучающихся по образовательным программам высшего образования: программам бакалавриата, программам специалитета, программам магистратуры и Положением о балльно-рейтинговой системе учета и оценки достижений обучающихся.

Наличие в Университете электронной информационно-образовательной среды, а также электронных образовательных ресурсов позволяет осваивать курс инвалидам и лицам с ОВЗ.

10.2. Особенности освоения учебной дисциплины инвалидами и лицами с ограниченными возможностями здоровья.

Особенности освоения учебной дисциплины инвалидами и лицами с ОВЗ определены в Положении об организации обучения инвалидов и лиц с ограниченными возможностями здоровья, утвержденном приказом ректора от «04» сентября 2017 года № 81-5.

Обучение инвалидов и лиц с ОВЗ может осуществляться индивидуально, а также с применением электронного обучения, дистанционных образовательных технологий.

Выбор методов и средств обучения, образовательных технологий и учебно-методического обеспечения реализации образовательной программы осуществляется Университетом самостоятельно, исходя из необходимости достижения обучающимися планируемых результатов освоения образовательной программы, а также с учетом индивидуальных возможностей обучающихся из числа инвалидов и лиц с ОВЗ.

Форма проведения промежуточной аттестации для студентов-инвалидов и лиц с ОВЗ устанавливается с учетом индивидуальных психофизических особенностей (устно, письменно на бумаге, письменно на компьютере, в форме тестирования и т.п.). При необходимости инвалидам и лицам с ОВЗ предоставляется дополнительное время для подготовки ответа на зачете или экзамене.

В группах, в состав которых входят студенты с ОВЗ, с целью реализации индивидуального подхода, а также принципа индивидуализации и дифференциации, рекомендуется использовать технологию нелинейной конструкции учебных занятий, предусматривающую одновременное сочетание фронтальных, групповых и индивидуальных форм работы с различными категориями студентов, в т.ч. имеющих ОВЗ.

В случае наличия обучающихся с нарушением функций опорно-двигательного аппарата, зрения и слуха, они обеспечиваются необходимым оборудованием, имеющимся в Университете, а также предоставляемым в рамках Соглашения с РУМЦ РГСУ от 12 января 2022г. №42-03/22.

11. Методические рекомендации преподавателю по организации учебного процесса по дисциплине

11.1. Преподавание учебной дисциплины осуществляется в соответствии с Федеральными государственными образовательными стандартами высшего образования, с учетом компетентностного подхода к обучению студентов.

При изучении дисциплины рекомендуется использовать следующий набор средств и способов обучения:

– рекомендуемую основную и дополнительную литературу;

– задания для подготовки к занятиям семинарского типа (вопросы для обсуждения, кейс задания, расчетные задачи и др.);

– задания для текущего контроля успеваемости (задания для самостоятельной работы обучающихся, тестовые задания в рамках электронной системы тестирования);

– вопросы и задания для подготовки к промежуточной аттестации по итогам освоения дисциплины, позволяющие оценить знания, умения и уровень приобретенных компетенций.

При проведении занятий лекционного и семинарского типа, в том числе в форме вебинаров и on-line курсов необходимо строго придерживаться тематического плана дисциплины, приведенного в РПД. Необходимо уделить внимание рассмотрению вопросов и заданий, включенных в тестовые оценочные задания, при необходимости, решить аналогичные задачи с объяснением алгоритма решения.

Следует обратить внимание обучающихся на то, что для успешной подготовки к текущему контролю (выполнению ТОЗ) и промежуточной аттестации (зачету или экзамену) недостаточно прочитать рабочий учебник, размещенный в личном кабинете. Нужно изучить материалы основной и дополнительной литературы, список которой приведен в РПД, законодательные и нормативные акты, а также материалы, рекомендованные в разделе «Перечень ресурсов информационно-телекоммуникационной сети «Интернет», необходимых для освоения дисциплины».

Текущий контроль успеваемости по учебной дисциплине и промежуточная аттестация осуществляются в соответствие с Положением о текущем контроле успеваемости и промежуточной аттестации обучающихся по образовательным программам высшего образования: программам бакалавриата, программам специалитета, программам магистратуры и Положением о балльно-рейтинговой системе учета и оценки достижений обучающихся. С основными принципами организации балльно-рейтинговой оценки достижений обучающихся, принятой в Университете, необходимо ознакомить на первом занятии.

11.2. Инновационные формы учебных занятий

При проведении учебных занятий обеспечивается развитие у обучающихся навыков командной работы, межличностной коммуникации, принятия решений, развитие лидерских качеств на основе инновационных (интерактивных) занятий: групповых дискуссий, ролевых игр, тренингов, анализа ситуаций и имитационных моделей, преподавания дисциплин в форме курсов, составленных на основе результатов научных исследований, проводимых Университетом, в том числе с учетом региональных особенностей профессиональной деятельности выпускников и потребностей работодателей) и т.п.

11.3.Инновационные образовательные технологии, используемые на занятиях лекционного и семинарского типа

Вид занятия	Используемые интерактивные образовательные технологии
Занятие лекционного типа	Не предусмотрено.
Семинарские и практические занятия	Не предусмотрено.

12. Перечень информационных технологий

Образовательный процесс по дисциплине поддерживается средствами электронной информационно-образовательной среды Университета, которая обеспечивает:

- доступ к учебным планам, рабочим программам дисциплин (модулей), практик, к изданиям электронных библиотечных систем и электронным образовательным ресурсам, указанным в рабочей программе, через личный кабинет студента и преподавателя;

- фиксацию хода образовательного процесса, результатов промежуточной аттестации и результатов освоения основной образовательной программы;

- проведение всех видов занятий, процедур оценки результатов обучения, реализация которых предусмотрена с применением дистанционных образовательных технологий;

- формирование электронного портфолио обучающегося, в том числе сохранение работ обучающегося, рецензий и оценок на эти работы со стороны любых участников образовательного процесса;

- взаимодействие между участниками образовательного процесса, в том числе синхронное и (или) асинхронное взаимодействие посредством сети Интернет.

Каждый обучающийся обеспечен индивидуальным неограниченным доступом к электронно-библиотечной системе (ЭБС университета), содержащей издания учебной, учебно-методической и иной литературы по основным изучаемым дисциплинам и сформированной на основании прямых договоров с правообладателями.

Перечень программного обеспечения определяется в п.13 РПД.

Профессиональные базы данных:
1. http://www.consultant.ru, Справочная правовая система «Консультант Плюс»
2. https://its.1c.ru, Профессиональная база ресурсов для 1С:Предприятие

13. Материально-техническая база

№ п/п	Наименование оборудованных учебных кабинетов, лабораторий	Перечень программного обеспечения
1.	Учебные аудитории для проведения занятий лекционного типа	1. Notepadqq (GNU GPL 3) 2. OnlyOffice (AGPLv3) 3. Remmina remote desktop client (free, Open Source) 4. Консультант+ (Коммерческая лицензия) 5. Mozilla Firefox (MPL 2)
2.	Компьютерные классы	1. 1С Предприятие 2. OnlyOffice (AGPLv3) 3. GIMP (GNU GPL 3) 4. Krita (GNU GPL 3) 5. Inkscape (GNU GPL 3) 6. SciLab (CeCILL) 7. Kaspersky Endpoint Security 8. Агент администрирования Kaspersky Security Center 9. Mozilla Firefox (MPL 2) 10. Apache NetBeans IDE (Apache License 2) 11. Notepadqq (GNU GPL 3) 12. VSCodium (MIT) 13. Mono Development IDE (MIT) 14. PyCharm Community Edition (Apache License 2) 15. Android Studio (Apache License 2) 16. Python (Python Software Foundation License) 17. Java (GNU GPL) 18. Node.js (MIT) 19. Git (GNU GPL 2) 20. GitHub Destkop (No Copyright) 21. UnityHub (Unity Companion License) 22. FreeCAD (LGPL-2.0-or-later) 23. Sweet Home 3D (GNU GPL 2+) 24. draw.io (Apache License 2) 25. DBeaver (Apache License 2) 26. PostgreSQL (License (free and open-source)) 27. MariaDB (GNU GPL) 28. PSPP (GNU GPL 3) 29. Wireshark (GNU GPL 2+) 30. Imunes (Creative Commons Attribution 4.0 International Public License) 31. VirtualBox (GNU GPL 2) 32.Apache (Apache License 2) 33. Консультант+ (Коммерческая лицензия) 34. OBS (GNU GPL)

3.	Учебные аудитории для проведения занятий семинарского типа, курсового проектирования (выполнения курсовых работ), групповых и индивидуальных консультаций, текущего контроля и промежуточной аттестации, а также самостоятельной работы обучающихся	1. 1С Предприятие 2. OnlyOffice (AGPLv3) 3. GIMP (GNU GPL 3) 4. Krita (GNU GPL 3) 5. Inkscape (GNU GPL 3) 6. SciLab (CeCILL) 7. Kaspersky Endpoint Security 8. Агент администрирования Kaspersky Security Center 9. Mozilla Firefox (MPL 2) 10. Apache NetBeans IDE (Apache License 2) 11. Notepadqq (GNU GPL 3) 12. VSCodium (MIT) 13. Mono Development IDE (MIT) 14. PyCharm Community Edition (Apache License 2) 15. Android Studio (Apache License 2) 16. Python (Python Software Foundation License) 17. Java (GNU GPL) 18. Node.js (MIT) 19. Git (GNU GPL 2) 20. GitHub Destkop (No Copyright) 21. UnityHub (Unity Companion License) 22. FreeCAD (LGPL-2.0-or-later) 23. Sweet Home 3D (GNU GPL 2+) 24. draw.io (Apache License 2) 25. DBeaver (Apache License 2) 26. PostgreSQL (License (free and open-source)) 27. MariaDB (GNU GPL) 28. PSPP (GNU GPL 3) 29. Wireshark (GNU GPL 2+) 30. Imunes (Creative Commons Attribution 4.0 International Public License) 31. VirtualBox (GNU GPL 2) 32.Apache (Apache License 2) 33. Консультант+ (Коммерческая лицензия) 34. OBS (GNU GPL)
4.	Помещение для самостоятельной работы / библиотека, читальный зал	1. 1С Предприятие 2. OnlyOffice (AGPLv3) 3. GIMP (GNU GPL 3) 4. Krita (GNU GPL 3) 5. Inkscape (GNU GPL 3) 6. SciLab (CeCILL) 7. Kaspersky Endpoint Security 8. Агент администрирования Kaspersky Security Center 9. Mozilla Firefox (MPL 2) 10. Apache NetBeans IDE (Apache License 2) 11. Notepadqq (GNU GPL 3) 12. VSCodium (MIT) 13. Mono Development IDE (MIT) 14. PyCharm Community Edition (Apache License 2)

		15. Android Studio (Apache License 2) 16. Python (Python Software Foundation License) 17. Java (GNU GPL) 18. Node.js (MIT) 19. Git (GNU GPL 2) 20. GitHub Destkop (No Copyright) 21. UnityHub (Unity Companion License) 22. FreeCAD (LGPL-2.0-or-later) 23. Sweet Home 3D (GNU GPL 2+) 24. draw.io (Apache License 2) 25. DBeaver (Apache License 2) 26. PostgreSQL (License (free and open-source)) 27. MariaDB (GNU GPL) 28. PSPP (GNU GPL 3) 29. Wireshark (GNU GPL 2+) 30. Imunes (Creative Commons Attribution 4.0 International Public License) 31. VirtualBox (GNU GPL 2) 32.Apache (Apache License 2) 33. Консультант+ (Коммерческая лицензия) 34. OBS (GNU GPL)

14. Сведения о разработчиках

Разработчик рабочей программы дисциплины: Королькова Ирина Анатольевна; к.т.н., доцент Зайцев Сергей Александрович;

15. Сведения об утверждении и внесении изменений в РПД

Рабочая программа дисциплины «Высокоуровневые методы программирования» составлена в соответствии с требованиями ФГОС ВО, рассмотрена и одобрена на заседании кафедры «Кафедра информационных систем» протокол № 6 от 18.12.2024 г.

Лист обновления (актуализации) рабочей программы дисциплины

Внесенные изменения	**Основание**
1) Внесены изменения в состав рекомендуемой литературы 2) Актуализирован перечень профессиональных баз данных и интернет-ресурсов 3) Актуализирована материально-техническая база 4) Актуализированы оценочные средства в рабочих программах дисциплин	Решение Ученого совета протокол № 6 от 21.03.2024 г.

Приложение 1
к рабочей программе по дисциплине
«Высокоуровневые методы программирования»

МУ ИВ МОСКОВСКИЙ УНИВЕРСИТЕТ ИМЕНИ С.Ю. ВИТТЕ

Оценочные средства по дисциплине Высокоуровневые методы программирования

Направление подготовки:
09.03.03 «Прикладная информатика»

Направленность (профиль) программы:
Искусственный интеллект и анализ данных

Уровень высшего образования:
бакалавриат

Шифр модуля	Наименование модуля
09.03.03:ВысМетПрог:1	Высокоуровневые методы программирования1 / Объектно-ориентированное программирование
09.03.03:ВысМетПрог:2	Высокоуровневые методы программирования2 / Функциональное программирование
09.03.03:ВысМетПрог:3	Высокоуровневые методы программирования3 / Создание графического интерфейса пользователя
09.03.03:ВысМетПрог:4	Высокоуровневые методы программирования4 / Web-программирование
09.03.03:ВысМетПрог:5	Высокоуровневые методы программирования5 / Программирование сетевых приложений
09.03.03:ВысМетПрог:6	Высокоуровневые методы программирования6 / Многопоточное программирование
09.03.03:ВысМетПрог:7	Высокоуровневые методы программирования7 / Программирование запросов к базе данных из Python
09.03.03:ВысМетПрог:8	Высокоуровневые методы программирования8 / Расширенные возможности работы с файлами

СОДЕРЖАНИЕ

1. Перечень компетенций, формируемых в процессе освоения учебной дисциплины.

1.1. Планируемые результаты освоения образовательной программы:

Коды компетенций	Содержание компетенций
ПК-1	- Способность разработки прикладного программного обеспечения, автоматизации работы с базами данных и документами, программирования бизнес-логики приложений, интеграции разнородных данных

1.2. Взаимосвязь планируемых результатов обучения по дисциплине с планируемыми результатами освоения образовательной программы

Коды компетенций ОПОП	Индикаторы	Знать	Уметь	Владеть
ПК-1	ПК-1.1. Знает технологии программирования прикладного программного обеспечения и бизнес-логики приложений	- технологии программирования с использованием языков высокого уровня; - технологии распределенных вычислений	- использовать языки высокого уровня для разработки прикладного программного обеспечения	- методами решения типовых алгоритмических задач
	ПК-1.2. Умеет разрабатывать и конфигурировать прикладное программное обеспечение	- последовательность разработки прикладного программного обеспечения	- использовать языки программирования высокого уровня для разработки многопоточных приложений	- методами передачи данных между потоками
	ПК-1.3. Владеет навыками автоматизации решения типовых задач, работы с базами данных и документами, интеграции разнородных данных в корпоративных информационных системах	- методы решения типовых алгоритмических задач; - методы проектирования интерфейса на языке Python с использованием библиотек визуализации	- программировать запросы к базе данных; - программировать динамически создаваемый и обновляемый интерфейс на языке Python	- методами доступа к данным с помощью стандартных интерфейсов подключения к данным; - навыками доступа к данным с помощью стандартных интерфейсов подключения

2. Описание показателей и критериев оценивания компетенций, описание шкал оценивания

2.1. Текущий контроль успеваемости по учебной дисциплине и промежуточная аттестация осуществляются в соответствие с Положением о текущем контроле успеваемости и промежуточной аттестации обучающихся по образовательным программам высшего образования: программам бакалавриата, программам специалитета, программам магистратуры и Положением о балльно-рейтинговой системе учета и оценки достижений обучающихся.

2.2. В соответствии с Положением о балльно-рейтинговой системе учета и оценки достижений обучающихся степень освоения компетенций оценивается по 100-балльной шкале, которая переводится в традиционную четырёхбалльную систему.

2.3. В ходе текущего контроля успеваемости при ответах на семинарских и практических занятиях, промежуточной аттестации в форме экзамена (зачет с оценкой) обучающиеся оцениваются по четырёхбалльной шкале: «отлично», «хорошо», «удовлетворительно», «неудовлетворительно»

- оценка «отлично» выставляется обучающимся, показавшим всестороннее, систематическое и глубокое знание учебно-программного материала, умение свободно выполнять задания, предусмотренные программой, усвоивших основную и дополнительную литературу, рекомендованную программой. Как правило, оценка «отлично» выставляется студентам, усвоившим взаимосвязь основных понятий дисциплины в их значении для приобретаемой профессии, проявившим творческие способности в понимании, изложении и использовании учебно-программного материала.

- оценка «хорошо» выставляется обучающимся, показавшим полное знание учебно-программного материала, успешно выполняющим предусмотренные в программе задания, усвоившим основную литературу, рекомендованную в программе. Как правило, оценка «хорошо» выставляется студентам, продемонстрировавшим систематический характер знаний по дисциплине и способным к их самостоятельному пополнению и обновлению в ходе дальнейшей учебной работы и профессиональной деятельности.

- оценка «удовлетворительно» выставляется обучающимся, показавшим знания основного учебно-программного материала в объеме, необходимом для дальнейшей учебы и предстоящей работы по специальности, справившимся с выполнением заданий, предусмотренных программой, ориентирующимся в основной литературе, рекомендованной программой. Как правило, оценка «удовлетворительно» выставляется студентам, допустившим погрешности в ответе на экзамене и при выполнении экзаменационных заданий, но обладающим необходимыми знаниями для их устранения под руководством преподавателя.

- оценка «неудовлетворительно» выставляется обучающимся, имеющим пробелы в знаниях основного учебно-программного материала, допустившим принципиальные ошибки в выполнении предусмотренных программой заданий. Как правило, оценка «неудовлетворительно» ставится студентам, которые не могут продолжить обучение или приступить к профессиональной деятельности по окончании вуза без дополнительных занятий по соответствующей дисциплине.

2.4. В ходе промежуточной аттестации в форме зачёта обучающиеся оцениваются «зачтено» или «не зачтено»:

- оценка «зачтено» выставляется обучающимся, показавшим знания основного учебно-программного материала, справившимся с выполнением заданий, предусмотренных программой, ориентирующимся в основной и дополнительной литературе, рекомендованной программой.

- оценка «не зачтено» выставляется обучающимся, имеющим пробелы в знаниях основного учебно-программного материала, допустившим принципиальные ошибки в выполнении предусмотренных программой заданий.

3. Типовые контрольные задания или иные материалы, необходимые для оценки результатов обучения по учебной дисциплине.

3.1. Примерные варианты тестовых оценочных заданий (ТОЗ) для контрольного рубежа в рамках текущего контроля

Тестовое задание	Оцениваемые индикаторы
S: Адаптер — это? +: SQL-объект, который переводит тип данных из Python в тип данных из SQL -: SQL-объект, который переводит тип данных из SQL в тип данных из Python -: SQL-объект, который оптимизирует SQL-запросы S: Что такое синхронизация? -: Организация абсолютно параллельной работы потоков на разных ядрах процессора +: Организация строго поочередной работы потоков над одним и тем же куском кода -: Организация работы потоков таким образом, чтобы они закончили выполняться в одно и то же время -: Процесс распределения одной и той же задачи между несколькими потоками S: Что принимает на вход конструктор класса socket? -: два целых числа -: строку и число -: число и строку +: кортеж S: Как правильно расшифровывается SMTP? -: Serial Mail Transit Protocol -: Serial Mail Transfer Protocol -: Simple Mail Transit Protocol +: Simple Mail Transfer Protocol S: Какой операции нет в SQL? +: CHECK -: SELECT	ПК-1.1; ПК-1.2; ПК-1.3

-: UPDATE
-: INSERT

S: Свойство bg класса Button определяет

-: цвет текста
-: цвет фона во время нажатия
-: цвет текста во время нажатия
+: цвет фона

S: Что произойдет если нажать сочетание клавиш Shift-Up (в поле предварительно набрали текст)

```
def selectAll(event):

    root.after(10, select_all, event.widget)

def select_all(widget):

    widget.selection_range(0, END)

    widget.icursor(END)

ent = Entry(width=40)

ent.focus_set()

ent.pack()

ent.bind('<Shift-Up>',selectAll)
```

-: курсор переместится вверх
+: будет выделен весь текст
-: фокус получит поле Entry
-: размер шрифта изменится с 40 на 10

S: Объектно-ориентированный язык:

+: Язык построенный на принципах ООП
+: В основе концепции лежит понятие объекта
+: Имеет особенности как наследование, полиморфизм, инкапсуляция
-: Имеет такие особенности как локализация, свойства, абстрагирование, и командные инструкции

S: AttributeError возникает:

+: При отсутствии атрибута класса
+: При отсутствии атрибута метода
+: При вызове приватной переменной
-: При вызове публичной переменной

S: Как называется программа для визуальной работы с базой данных? +: DB browser for SQLite -: SQLite Viewer -: Python SQL browser -: SQLite Manager S: Какой метод нужно переопределить, чтобы задать выполняемый код потока? +: run() -: start() -: __init__() -: exec() S: Создание атрибутов класса: +: Происходит на том же уровне вложенности что и определение методов -: Объявляется внутри переменной через self -: Объявляется через декораторов S: Основная концепция наследования заключается в том, что … - : +: абстрактный тип данных может наследовать данные и функциональность некоторого существующего типа -: размещение в одном компоненте данных и методов, которые с ними работают -: использование только тех характеристик объекта, которые с достаточной точностью представляют его в данной системе S: С помощью какой инструкции создаются классы в Python: -: Class Имя_класса +: class имя_класса: -: имя_класса class(){} -: class Имя_класса{}: S: Python является: +: Объектно-ориентированным языком программирования -: Структурным языком программирования -: Процедурным языком программирования -: Аспекта-ориентированным языком программирования	

.

3.2. Вопросы для подготовки к промежуточной аттестации (к зачету/экзамену)

1. Декомпозиция в Python.
2. Классы объектов встроенных типов в Python.
3. Вызовы конструкторов классов и методов-деструкторов классов в Python.
4. Инкапсуляция. Доступ к свойствам и атрибутам классов в Python.

5. Сокрытие данных атрибута класса в Python.

6. Полиморфизм в Python.

7. Имитация типов в Python.

8. Наследование в Python.

9. Множественное наследование от нескольких классов в Python.

10. Порядок разрешения методов (method resolution order) в Python.

11. Контейнеры в Python.

12. Итераторы в Python.

13. Ассоциации и циклические ссылки в Python.

14. Слабые ссылки в Python. Модуль weakref.

15. Статические методы для применения описателей (descriptors) в Python.

16. Методы класса в модуле tree пакета nltk.

17. Метаклассы в Python. Динамическое создание классов в функции-фабрике классов.

18. Создание собственных мультиметодов с помощью модуля Multimethod.

19. Функции преобразования типов и классы: coerce, str, repr, int, list, tuple, long, float, complex, dict, super, file, bool, object.

20. Числовые и строковые функции: abs, divmod, ord, pow, len, chr, unichr, hex, oct, cmp, round, unicode.

21. Функции обработки данных: apply, map, filter, reduce, zip, range, xrange, max, min, iter, enumerate, sum.

22. Функции определения свойств: hash, id, callable, issubclass, isinstance, type.

23. Функции для доступа к внутренним структурам: locals, globals, vars, intern, dir.

24. Функции компиляции и исполнения: eval, execfile, reload, __import__, compile.

25. Функции ввода-вывода: input, raw_input, open.

26. Функции для работы с атрибутами: getattr, setattr, delattr, hasattr.

27. Функции-"украшатели" методов классов: staticmethod, classmethod, property.

28. Прочие функции: buffer, slice.

29. Модули: sys, atexit, copy, traceback, math, cmath, random, time, calendar, datetime, sets, array, struct, itertools, locale, gettext.

30. Модули: pdb, hotshot, profile, unittest, pydoc. Пакеты docutils, distutils.

31. Модули: os, os.path, getopt, glob, popen2, shutil, select, signal, stat, tempfile.

32. Модули: pickle, shelve, anydbm, gdbm, gzip, zlib, zipfile, bz2, csv, tarfile.

33. Платформо-зависимые модули. Для UNIX: commands, pwd, grp, fcntl, resource, termios, readline, rlcompleter. Для Windows: msvcrt, _winreg, winsound.

34. Модули: cgi, Cookie, urllib, urlparse, httplib, smtplib, poplib, telnetlib, socket, asyncore. Примеры серверов: SocketServer, BaseHTTPServer, xmlrpclib, asynchat.

35. Модули: quopri, uu, base64, binhex, binascii, rfc822, mimetools, MimeWriter, multifile, mailbox. Пакет email.

36. Функция apply()

37. Функции range() и xrange()

38. Функция map()

39. Функция filter().

40. Функция sum().

41. Функция reduce().

42. Функция zip().

43. Функция iter().

44. Функция enumerate().

45. Функция sorted().
46. Функция itertools.chain().
47. Функция itertools.repeat().
48. Функция itertools.count().
49. Функция itertools.cycle().
50. Функции itertools.imap(), itertools.starmap() и itertools.ifilter().
51. Функции itertools.takewhile() и itertools.dropwhile().
52. Функция itertools.izip().
53. Функция itertools.groupby().
54. Функция itertools.tee().
55. Итератор, определенный пользователем.
56. Формирование функции-генератора.
57. Генераторное выражение.
58. Библиотека Xoltar toolkit с модулем functional.
59. Виджеты и их свойства.
60. Менеджер геометрии Pack.
61. Менеджер геометрии Place.
62. Менеджер геометрии Grid.
63. Text – многострочное текстовое поле.
64. Метод bind().
65. События.
66. Создание графических примитивов.
67. Canvas. Идентификаторы, теги и анимация.
68. CGI-сценарии.
69. Метод GET.
70. Метод POST.
71. Встроенный Web-сервер (ZServer).
72. Среда разработчика zope.
73. Поддержка языков сценариев Python, Perl и собственного DTML.
74. Модули для web-сервера mod_python.
75. Flask.
76. Django Framework.
77. Модуль socket.
78. Функция socket.getservbyname().
79. Модуль smtplib.
80. Конструктор класса POP3 из модуля poplib.
81. Модуль poplib.
82. Функция urllib.urlopen().
83. Функции urllib.urlencode().
84. Модуль urllib.
85. Функция urlretrieve().
86. Функция quote(s, safe='/').
87. Функциональность модуля urllib2.
88. Виды многопоточности.
89. Запуск и завершение потоков.
90. Состояния потоков.
91. Приоритеты.
92. Блокировки.

93. Функции модуля threading.
94. Конструктор класса threading.Thread.
95. Методы управления жизнью потоков.
96. Класс threading.Timer.
97. Передача данных между потоками.
98. Синхронизация.
99. Замки. Объект класса Lock.
100. Тупиковая ситуация (deadlock).
101. Семафоры. Конструктор класса threading.Semaphore.
102. События.
103. Экземпляры класса threading.Event.
104. Условия. Экземпляры класса threading.Condition.
105. Очередь. Модуль Queue.
106. Модуль thread.
107. Методы объекта-соединения.
108. Функция connect().
109. Атрибуты объекта-курсора.
110. Названия для объектов-типов, используемых для описания полей базы данных.
111. Реляционная база данных.
112. Подключение к базе данных.
113. Создание одного или нескольких курсоров (вызов метода объекта-соединения cursor() с получением объекта-курсора).
114. Исполнение команды или запроса (вызов метода execute() или его вариантов).
115. Получение результатов запроса (вызов метода fetchone() или его вариантов).
116. Завершение транзакции или ее откат (вызов метода объекта-соединения commit() или rollback()).
117. Закрытие подключения методом close().
118. Сценарии SQL для создания таблиц (CREATE TABLE).
119. Сценарии SQL для наполнения таблиц (INSERT).
120. Сценарии SQL для выборки данных (SELECT).
121. Менеджер контекста with.
122. Запись в файл с помощью print.
123. Чтение из текстового файла.
124. Двоичные файлы и тип bytes.
125. Модуль pickle.
126. Модуль struct.
127. Формат CSV.
128. Функция reader(csvfile[, dialect='excel'[, fmtparam]]).
129. Функция writer(csvfile[, dialect='excel'[, fmtparam]]).
130. Последовательная обработка и работа с объектной моделью документа.
131. Формирование XML-документа.
132. XML-анализаторы. Пакет PyXML.
133. Как создать поток в Python, используя стандартный модуль threading?
134. Как обеспечить синхронизацию потоков в Python, чтобы избежать проблем с доступом к общим данным?

135. Какие проблемы могут возникнуть при использовании глобальных переменных в многопоточных приложениях, и как их можно решить?

136. Какие методы блокировки предоставляет модуль threading, и в каких случаях их следует использовать?

137. Как реализовать передачу данных между потоками в Python? Приведите пример использования очереди

138. Какие существуют подходы к реализации параллельного выполнения кода в Python, помимо использования потоков?

139. Какие существуют подходы к обработке исключений в многопоточных программах на Python? Как избежать возможных проблем при обработке ошибок?

4. Методические материалы, определяющие процедуры оценивания результатов обучения по учебной дисциплине.

Процедура оценивания результатов обучения по учебной дисциплине осуществляется на основе балльно-рейтинговой системы, в соответствии с Положением о балльно-рейтинговой системе оценки достижений обучающихся, а также Положением о текущем контроле и промежуточной аттестации обучающихся, утвержденными приказом ректора.

4.1 Первый этап: Проведение текущего контроля успеваемости по дисциплине

Проведение текущего контроля успеваемости по дисциплине осуществляется в ходе контактной работы с преподавателем в рамках аудиторных занятий и в ходе самостоятельной работы студента.

Текущий контроль в ходе контактной работы осуществляется по следующим видам:

1) Вид контроля: проверка сформированности компетенций в ходе самостоятельной работы обучающихся; текущий опрос, проводимый во время аудиторных (семинарских/практических/лабораторных) занятий; оценивание подготовленных докладов, сообщений, презентаций, домашних заданий.

Порядок проведения: в ходе подготовки к занятиям оценивается выполнение задания, рекомендованного к самостоятельной работе обучающихся, путем выборочной проверки.

Фиксируются результаты работы студентов в ходе проведения семинарских и практических занятий (активность, полнота ответов, способность поддерживать дискуссию, профессиональный язык и др.).

В ходе отдельных занятий обеспечивается проведение письменных опросов по тематике прошедших занятий. В ходе выполнения заданий обучающийся должен в меру имеющихся знаний, умений, навыков, сформированности компетенции дать развернутые ответы на поставленные в задании открытые вопросы и ответить на вопросы закрытого типа в установленное преподавателем время. Продолжительность проведения процедуры определяется преподавателем самостоятельно, исходя из сложности индивидуальных заданий, количества вопросов, объема оцениваемого учебного материала.

Задания по подготовке докладов, сообщений, презентаций, домашних заданий выдаются заранее при подготовке к семинарских и практическим занятиям; подготовленные работы оцениваются с фиксацией в журнале учета посещаемости и успеваемости обучающихся.

2) Вид контроля: Контроль с использованием тестовых оценочных заданий по итогам освоения модулей дисциплины (Рубежный контроль (РК)).

Порядок проведения: До начала проведения процедуры преподавателем подготавливаются необходимые оценочные материалы для оценки знаний, умений, навыков.

Оценка знаний, умений и навыков, характеризующих сформированность компетенций, осуществляется с помощью тестовых оценочных заданий (ТОЗ).

ТОЗ включают в себя три группы заданий.

Задания А (тесты закрытой формы) – задания с выбором правильного ответа. Эти задания представляются в трех вариантах:

- задания, которые имеют один правильный и остальные неправильные (задания с выбором одного правильного ответа);

- задания с выбором нескольких правильных ответов.

Задания В (тесты открытой формы) – задания без готового ответа. Эти задания также представляются в трех вариантах:

- задания в открытой форме, когда испытуемому во время тестирования ответ необходимо вписать самому, в отведенном для этого месте;

- задания, где элементам одного множества требуется поставить в соответствие элементы другого множества (задания на установление соответствия);

- задания на установление правильной последовательности вычислений, действий, операций, терминов в определениях понятий (задания на установление правильной последовательности).

Задания С – кейс-задания или практические задачи. Эти задания представлены в двух вариантах (также возможно их сочетание):

- расчетные задания содержат краткое и точное изложение ситуации с конкретными цифрами и данными. Для такого типа заданий существует определенное количество (или один) правильных ответов. Задания предназначены для оценки умения студента использовать в конкретной ситуации формулы, закономерности, технологии в определенной области знаний;

- логико-аналитические задания, которые представляют собой материал с большим количеством данных и предназначены для оценки логики мышления, умения анализировать представленные ситуации и направлены на формирование навыков профессиональной деятельности (в профессиональной области). Такие задания предполагают формулирование подвопросов, которые предусматривают выбор из нескольких вариантов ответов (по типу заданий А и В). Общее количество подвопросов к каждому такому заданию равно пяти.

Внеаудиторная контактная работа преподавателя с обучающимся осуществляется в ходе выполнения рейтинговой работы и контроля со стороны преподавателя за самостоятельной работой студента. Текущей контроль в ходе самостоятельной работы осуществляется в следующем виде:

3) Вид контроля: Подготовка курсовой (рейтинговой) работы (при наличии в учебном плане).

Контролируемые компетенции: ПК-1

Технология проведения: За каждым обучающимся, принимающим участие в процедуре преподавателем закрепляется тема курсовой (рейтинговой) работы. После получения задания и в процессе его подготовки обучающийся должен в меру имеющихся знаний, умений, навыков, сформированности компетенции дать развернутое раскрытие темы, выполнить расчетное или иное задание.

4.2 Второй этап: Проведение промежуточной аттестации по учебной дисциплине.

В соответствие с базовым учебным планом по учебной дисциплине предусмотрена подготовка и сдача экзамена или (и) зачета.

Порядок проведения промежуточной аттестации регламентируется Положением о текущем контроле и промежуточной аттестации, утвержденным приказом ректора Университета.