



МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ
ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение
высшего профессионального образования
«Дальневосточный федеральный университет»
(ДФУ)

ШКОЛА ЕСТЕСТВЕННЫХ НАУК

«СОГЛАСОВАНО»

Руководитель образовательной программы

А.С. Величко

«09» июля 2015 г.

«УТВЕРЖДАЮ»

Врио заведующего кафедрой
математических методов в экономике

А.С. Величко

«09» июля 2015 г.

РАБОЧАЯ ПРОГРАММА УЧЕБНОЙ ДИСЦИПЛИНЫ
Операционные системы, сети, компьютерная графика
Направление подготовки 01.03.04 Прикладная математика
профиль «Математические методы в экономике»
Форма подготовки очная

курс 3 семестр 5, 6
лекции 36 час.
практические занятия 36 час.
лабораторные работы 18 час.
в том числе с использованием МАО лек. 36 час. / пр. 36 час. / лаб. 18 час.
всего часов аудиторной нагрузки 90 час.
в том числе с использованием МАО 90 час.
самостоятельная работа 90 час.
в том числе на подготовку к экзамену 27 час.
контрольные работы (количество) не предусмотрены
курсовая работа / курсовой проект не предусмотрены
зачет 5 семестр
экзамен 6 семестр

Рабочая программа составлена в соответствии с требованиями федерального государственного образовательного стандарта высшего образования, утвержденного приказом Министерства образования и науки РФ от 12.03.2015 № 208

Рабочая программа обсуждена на заседании кафедры математических методов в экономике, протокол № 17 от «09» июля 2015 г.

Врио заведующего кафедрой математических методов в экономике, к.ф.-м.н., доцент А.С. Величко

Составитель:

старший преподаватель кафедры математических методов в экономике Е.А. Воронцова

АННОТАЦИЯ

Дисциплина «Операционные системы, сети, компьютерная графика» предназначена для студентов направления подготовки 01.03.04 «Прикладная математика», профиль «Математические методы в экономике».

Общая трудоемкость освоения дисциплины составляет 5 зачётных единиц (180 часов). Дисциплина реализуется на 3 курсе в 5-м и в 6-м семестрах. Дисциплина входит в обязательные дисциплины базовой части блока «Дисциплины (модули)».

Особенности построения курса: лекции (36 часов), лабораторные работы (18 часов), практические занятия (36 часов), самостоятельная работа (63 часа), подготовка к экзамену (27 часов).

Содержание дисциплины охватывает следующий круг вопросов: операционные системы, язык программирования Python, сети, компьютерная графика.

Цель - знакомство студентов с основами организации операционных систем, изучение принципов функционирования операционных систем, получение практических навыков по написанию программ на языке программирования Python; изучение сетевых структур и топологий; получение практических навыков по графической компьютерной визуализации научных данных и их предварительной обработки, в том числе с помощью методов машинного обучения.

Задачи:

- развитие способности осваивать современные технологии программирования на языке программирования Python;
- развитие способность демонстрировать знание основ организации операционных систем, сетевых структур и топологии сетей;
- развитие способности разрабатывать программы, использующие возможности операционных систем;
- развитие способности отлаживать и тестировать прикладное программное обеспечение;

- развитие способности настраивать, тестировать и осуществлять проверку вычислительной техники и программных средств;
- развитие готовности настраивать, тестировать и осуществлять проверку вычислительной техники и программных средств;
- развитие способности демонстрировать знания современных языков программирования;
- развитие готовности демонстрировать знания современных языков программирования.

Для успешного изучения дисциплины «Операционные системы, сети, компьютерная графика» у обучающихся должны быть сформированы следующие предварительные компетенции:

- способность выявить сущность проблем, возникающих в ходе профессиональной деятельности, готовность использовать для их решения соответствующий аппарат;
- способность использовать для работы операционную систему ЭВМ Microsoft Windows, информационно-телекоммуникационную сеть "Интернет"
- знание одного из современных компилируемых языков программирования (например, C++);
- умение работать в операционной системе Linux.

Уровень подачи материала курса в достаточной степени опирается на следующие предметы, изученные студентами на 1 и 2 курсах: программные и аппаратные средства информатики, программирование для ЭВМ, модели и методы прикладной математики, теория вероятности и математическая статистика, инструментальные средства обработки и управления данными.

В результате изучения данной дисциплины у обучающихся формируются следующие общепрофессиональные, профессиональные компетенции (элементы компетенций).

Код и формулировка компетенции	Этапы формирования компетенции	
ОПК-2 - способность использовать современные математические методы и современные прикладные программные средства и осваивать современные технологии программирования	Знает	современные технологии программирования на языке Python; современные математические методы машинного обучения
	Умеет	разрабатывать программные продукты с помощью современных технологий программирования
	Владеет	эффективными методами разработки программных продуктов с помощью современных технологий программирования
ПК-2 - способность и готовность настраивать, тестировать и осуществлять проверку вычислительной техники и программных средств	Знает	способы тестирования программных средств, в том числе, разработанных самостоятельно
	Умеет	находить и исправлять ошибки в тестируемых программных продуктах
	Владеет	навыками настройки вычислительной техники для работы в интегрированной среде разработки, навыками тестирования и проверки программных продуктов
ПК-3 – способность и готовность демонстрировать знания современных языков программирования, операционных систем, офисных приложений, информационно-телекоммуникационной сети "Интернет", способов и механизмов управления данными, принципов организации, состава и схемы работы операционных систем	Знает	принципы работы современных операционных систем, современные языки программирования (в частности, язык Python и его библиотеки NumPy, SciPy, Matplotlib, Pandas, Sklearn)
	Умеет	применять знания современных языков программирования и принципов работы современных операционных систем для разработки программных продуктов
	Владеет	навыками разработки программ и реализации методы обработки научных данных на современных языках программирования

Для формирования вышеуказанных компетенций в рамках дисциплины «Операционные системы, сети, компьютерная графика» применяются следующие методы активного/интерактивного обучения: лекция-беседа, занятие-дискуссия, выполнение проектов при создании программного обеспечения с использованием компьютерных технологий.

I. СТРУКТУРА И СОДЕРЖАНИЕ ТЕОРЕТИЧЕСКОЙ ЧАСТИ КУРСА

**Лекции (36 часов, в том числе 36 часов с использованием методов
активного обучения)**

МОДУЛЬ 1 (18 ЧАСОВ, В ТОМ ЧИСЛЕ 18 ЧАСОВ С ИСПОЛЬЗОВАНИЕМ МЕТОДОВ АКТИВНОГО ОБУЧЕНИЯ)

Раздел I. Архитектура компьютерной системы (10 часов)

Тема 1. Основные понятия архитектуры (4 часа)

Управление прерываниями, памятью, вводом-выводом. Иерархия памяти. Кэширование. Защита памяти. Аппаратная защита памяти в системах с теговой архитектурой.

Тема 2. Понятие операционной системы (ОС) (2 часа)

Определение ОС. Функции ОС. История ОС. Отечественные ОС. ОС UNIX. Режимы пакетной обработки, мультипрограммирования, разделения времени. Особенности ОС для различных классов компьютерных систем (многопроцессорные и распределенные системы, настольные, карманные и др.). ОС реального времени.

Тема 3. Процессы и потоки (2 часа)

Основные сведения о процессах. Модели процесса. Описание процессов. Выполнение кода операционной системы по отношению к процессам. Потоки. Основные операции с потоками. Способы реализации потоков. Взаимодействие процессов.

Тема 4. Архитектура UNIX, Windows и MS-DOS (2 часа)

Архитектура ОС UNIX. Системные вызовы управления процессами в ОС UNIX. Сигналы в ОС UNIX. Архитектура ОС MS-DOS. Системы Windows.

Раздел II. Коммуникационные сети (4 часа)

Тема 5. Сети и сетевые структуры (2 часа)

Определение. Распределенные и сетевые системы. Топологии и типы сетей. Коммуникации по сети. Маршрутизация.

Тема 6. Коммуникационные протоколы (2 часа)

Сетевые протоколы. Уровни организации сетей согласно модели ISO/OSI. Протокол TCP/IP. Протоколы GPRS, EDGE, 3G, LTE, 802.11x (Wi-Fi). Безопасность. Сетевые и системные угрозы (атаки). Борьба с атаками. История подводного Интернета.

МОДУЛЬ 2 (18 ЧАСОВ, В ТОМ ЧИСЛЕ 18 ЧАСОВ С ИСПОЛЬЗОВАНИЕМ МЕТОДОВ АКТИВНОГО ОБУЧЕНИЯ)

Раздел I. Язык программирования Python и компьютерная графика (10 часов)

Тема 3. Язык программирования Python (8 часов)

Интерпретируемые языки программирования. Основные конструкции языка программирования Python. Синтаксис, семантика. Функции. Списки, словари, кортежи, множества, их особенности, правила использования и отличия. Библиотека для работы с массивами NumPy. Библиотека для научных вычислений SciPy.

Тема 4. Компьютерная графика (2 часа)

Библиотека для обработки данных Pandas. Компьютерная графическая визуализация данных в Matplotlib.

Раздел II. Машинное обучение (8 часов)

Тема 5. Методы и задачи машинного обучения (4 часа)

Распознавание спама в почтовых сообщениях. Предобработка данных в Pandas. Решающие деревья. Выбор метрики. Нормализация признаков.

Тема 6. Обработка данных с помощью машинного обучения (4 часа)

Метод опорных векторов. Анализ текстов. Логистическая регрессия.

II. СТРУКТУРА И СОДЕРЖАНИЕ ПРАКТИЧЕСКОЙ ЧАСТИ КУРСА

Практические занятия (36 часов, в том числе 36 часов с использованием методов активного обучения)

Модуль 1 (36 часов, в том числе 36 часов с использованием методов активного обучения)

Тема 1. Язык программирования Python. Задача о пенсионных накоплениях.

Задача «Диофантовы уравнения и Макдональдс» (4 часа)

Тема 2. Адаптивный тренажёр Python (8 часов)

Тема 3. Словари, кортежи, списки, множества (4 часа)

Тема 4. Работы с библиотекой NumPy (4 часа)

Тема 5. Работа с библиотекой SciPy (4 часа)

Тема 6. Компьютерная графика и Matplotlib (4 часа)

Тема 7. Защита проектов — приложений для анализа данных, использующих особенности операционной системы (8 часов)

Лабораторные работы (18 часов, в том числе 18 часов с использованием методов активного обучения)

Модуль 2 (18 часов, в том числе 18 часов с использованием методов активного обучения)

Лабораторная работа № 1. Предобработка данных в Pandas (4 часа, в том числе 4 часа с использованием методов активного обучения)

Лабораторная работа № 2. Важность признаков (2 часа, в том числе 2 часа с использованием методов активного обучения)

Лабораторная работа № 3. Выбор числа соседей (2 часа, в том числе 2 часа с использованием методов активного обучения)

Лабораторная работа № 4. Выбор метрики (2 часа, в том числе 2 часа с использованием методов активного обучения)

Лабораторная работа № 5. Нормализация признаков (2 часа, в том числе 8 часов с использованием методов активного обучения)

Лабораторная работа № 6. Метод опорных векторов (2 часа, в том числе 2 часа с использованием методов активного обучения)

Лабораторная работа № 7. Логистическая регрессия (4 часа, в том числе 4 часа с использованием методов активного обучения)

III. УЧЕБНО-МЕТОДИЧЕСКОЕ ОБЕСПЕЧЕНИЕ САМОСТОЯТЕЛЬНОЙ РАБОТЫ ОБУЧАЮЩИХСЯ

Учебно-методическое обеспечение самостоятельной работы обучающихся по дисциплине «Операционные системы, сети, компьютерная графика» представлено в Приложении 1 и включает в себя:

план-график выполнения самостоятельной работы по дисциплине, в том числе примерные нормы времени на выполнение по каждому заданию;

характеристика заданий для самостоятельной работы обучающихся и методические рекомендации по их выполнению;

требования к представлению и оформлению результатов самостоятельной работы;

критерии оценки выполнения самостоятельной работы.

IV. КОНТРОЛЬ ДОСТИЖЕНИЯ ЦЕЛЕЙ КУРСА

Контролируемые разделы дисциплины, этапы формирования компетенций, виды оценочных средств, зачетно-экзаменационные материалы, комплекты оценочных средств для текущей аттестации, описание показателей и критериев оценивания компетенций на различных этапах их формирования, методические материалы, определяющие процедуры оценивания знаний, умений и навыков и (или) опыта деятельности, а также критерии и показатели, необходимые для оценки знаний, умений, навыков и характеризующие этапы

формирования компетенций в процессе освоения образовательной программы, представлены в Приложении 2.

V. СПИСОК УЧЕБНОЙ ЛИТЕРАТУРЫ И ИНФОРМАЦИОННО-МЕТОДИЧЕСКОЕ ОБЕСПЕЧЕНИЕ ДИСЦИПЛИНЫ

Основная литература

(электронные и печатные издания)

1. Кручинин А.Ю. Операционные системы [Электронный ресурс] : учебное пособие. — Оренбург: изд-во «Бибком», 2009. — Режим доступа: <http://www.iprbookshop.ru/30115>.

2. Одиноков, В. В. Операционные системы и сети [Электронный ресурс] : учебное пособие. - Томск: Томский государственный университет систем управления и радиоэлектроники, 2007. - 391 с. — Режим доступа: <http://www.iprbookshop.ru/13951>.

3. Перемитина, Т. О. Компьютерная графика [Электронный ресурс] : учебное пособие. - Томск: изд-во «Эль Контент», Томский государственный университет систем управления и радиоэлектроники, 2012. - 144 с. — Режим доступа: <http://www.iprbookshop.ru/13940>.

Дополнительная литература

(печатные и электронные издания)

1. Маккинли У. Python и анализ данных [Электронный ресурс]. — М.: ДМК Пресс, 2015. — 482 с. — Режим доступа: http://e.lanbook.com/books/element.php?pl1_id=73074.

2. Саммерфилд М. Python на практике [Электронный ресурс]. — М.: ДМК Пресс, 2016. — 338 с. — Режим доступа: http://e.lanbook.com/books/element.php?pl1_id=66480.

3. Сузи Р.А. Язык программирования Python [Электронный ресурс]. М.: Интуит, 2007. — 328 с. — Режим доступа: <http://www.iprbookshop.ru/22450>.

**Перечень ресурсов информационно-телекоммуникационной сети
«Интернет»**

1. Linux. URL: <http://www.linux.org/>
2. The UNIX System. URL: <http://www.unix.org/>
3. Чем отличается устройство Windows - Microsoft Windows. URL: <http://windows.microsoft.com/ru-ru/windows-8/compare>
4. Онлайн-курс «Математика и Python для анализа данных». МФТИ, Академия Яндекса. <https://www.coursera.org/learn/mathematics-and-python>
5. Онлайн-курс «Введение в машинное обучение». Высшая школа экономики. <https://www.coursera.org/learn/vvedenie-mashinnoe-obuchenie>

**Перечень дополнительных информационно-методических
материалов**

1. Таненбаум Э. Современные операционные системы. 3-е издание. СПб.: Питер, 2010.
2. Илющечкин В.М. Операционные системы: учебное пособие. М.: Бином, Лаборатория знаний, 2011.
3. Бендел Д., Нейпир Р. Использование Linux. 6-е издание. М.: Вильямс, 2003.
4. Стивенс У. UNIX: взаимодействие процессов. СПб.: Питер, 2003.
5. Брейман А.Д. Сети ЭВМ и телекоммуникации. Глобальные сети. Учебное пособие. М.: МГУПИ, 2006.
6. Алиев Т.И. Сети ЭВМ и телекоммуникации. СПб.: СПбГУ ИТМО, 2011.
7. Silbershatz A., Galvin P. B., Gagne G. Operating System Concepts. 9th Edition International Student Version edition. John Wiley & Sons, 2013.
8. Цикритзис Д., Бернстайн Ф. Операционные системы. М.: Мир, 1977.

Перечень информационных технологий и программного обеспечения

При осуществлении образовательного процесса по дисциплине используется набор библиотек языка Python (Anaconda одной из последних версий), IPython Notebook. Обязателен компьютерный класс с доступом в Интернет.

VI. МЕТОДИЧЕСКИЕ УКАЗАНИЯ ПО ОСВОЕНИЮ ДИСЦИПЛИНЫ

Рекомендации по планированию и организации времени, отведенного на изучение дисциплины, описание последовательности действий обучающихся

Освоение дисциплины следует начинать с изучения рабочей учебной программы, которая содержит основные требования к знаниям, умениям и навыкам. Обязательно следует учитывать рекомендации преподавателя, данные в ходе установочных занятий. Затем – приступать к изучению отдельных разделов и тем в порядке, предусмотренном программой.

Получив представление об основном содержании раздела, темы, необходимо изучить материал с помощью рекомендуемой основной литературы. Целесообразно составить краткий конспект или схему, отображающую смысл и связи основных понятий данного раздела и включенных в него тем. Обязательно следует записывать возникшие вопросы, на которые не удалось ответить самостоятельно.

Подготовку к началу обучения включает несколько необходимых пунктов:

- 1) Необходимо создать для себя рациональный и эмоционально достаточный уровень мотивации к последовательному и планомерному изучению дисциплины.

- 2) Необходимо изучить список рекомендованной основной и дополнительной литературы и убедиться в её наличии у себя дома или в библиотеке в бумажном или электронном виде.

3) Необходимо иметь «под рукой» специальные и универсальные словари, справочники и энциклопедии, для того, чтобы постоянно уточнять значения используемых терминов и понятий. Пользование словарями и справочниками необходимо сделать привычкой. Опыт показывает, что неудовлетворительное усвоение предмета зачастую коренится в неточном, смутном или неправильном понимании и употреблении понятийного аппарата учебной дисциплины.

4) Желательно в самом начале периода обучения возможно тщательнее спланировать время, отводимое на работу с источниками и литературой по дисциплине, представить этот план в наглядной форме (график работы с датами) и в дальнейшем его придерживаться, не допуская срывов графика индивидуальной работы и «аврала» в предсессионный период. Пренебрежение этим пунктом приводит к переутомлению и резкому снижению качества усвоения учебного материала.

Рекомендации по работе с литературой

1) Всю учебную литературу желательно изучать «под конспект». Чтение литературы, не сопровождаемое конспектированием, даже пусть самым кратким – бесполезная работа. Цель написания конспекта по дисциплине – сформировать навыки по поиску, отбору, анализу и формулированию учебного материала. Эти навыки обязательны для любого специалиста с высшим образованием независимо от выбранной специальности.

2) Написание конспекта должно быть творческим – нужно не переписывать текст из источников, но пытаться кратко излагать своими словами содержание ответа, при этом максимально структурируя конспект, используя символы и условные обозначения. Копирование и «заучивание» неосмысленного текста трудоемко и по большому счету не имеет большой познавательной и практической ценности.

3) При написании конспекта используется тетрадь, поля в которой обязательны. Страницы нумеруются, каждый новый вопрос начинается с нового листа, для каждого экзаменационного вопроса отводится 1-2 страницы конспекта. На полях размещается вся вспомогательная информация – ссылки, вопросы, условные обозначения и т.д.

4) В итоге данной работы «идеальным» является полный конспект по программе дисциплины, с выделенными определениями, узловыми пунктами, примерами, неясными моментами, проставленными на полях вопросами.

5) При работе над конспектом обязательно выявляются и отмечаются трудные для самостоятельного изучения вопросы, с которыми уместно обратиться к преподавателю при посещении установочных лекций и консультаций, либо в индивидуальном порядке.

6) При чтении учебной и научной литературы всегда следить за точным и полным пониманием значения терминов и содержания понятий, используемых в тексте. Всегда следует уточнять значения по словарям или энциклопедиям, при необходимости записывать.

7) При написании учебного конспекта обязательно указывать все прорабатываемые источники, автор, название, дата и место издания, с указанием использованных страниц.

Подготовка к промежуточной аттестации по дисциплине: экзамену (зачету)

К аттестации допускаются студенты, которые систематически в течение всего семестра посещали и работали на занятиях и показали уверенные знания в ходе выполнении практических заданий и лабораторных работ.

Непосредственная подготовка к аттестации осуществляется по вопросам, представленным в рабочей учебной программе. Тщательно изучите формулировку каждого вопроса, вникните в его суть, составьте план ответа. Обычно план включает в себя:

— определение сущности рассматриваемого вопроса, основных

положений, утверждений, определение необходимости их доказательства;

— запись обозначений, формул, необходимых для полного раскрытия вопроса;

— графический материал (таблицы, рисунки, графики), необходимые для раскрытия сущности вопроса;

— роль и значение рассматриваемого материала для практической деятельности, примеры использования в практической деятельности.

VII. МАТЕРИАЛЬНО-ТЕХНИЧЕСКОЕ ОБЕСПЕЧЕНИЕ ДИСЦИПЛИНЫ

Для осуществления образовательного процесса по дисциплине необходим компьютерный класс с персональными компьютерами с доступом в сеть «Интернет» и установленным пакетом программ Anaconda (continuum.io) – полный набор библиотек для работы с языком программирования Python. В компьютерном классе должно быть оборудование мультимедийного типа (мультимедийный проектор, настенный экран) и доска.



МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ
ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение
высшего профессионального образования
«Дальневосточный федеральный университет»
(ДВФУ)

ШКОЛА ЕСТЕСТВЕННЫХ НАУК

**УЧЕБНО-МЕТОДИЧЕСКОЕ ОБЕСПЕЧЕНИЕ САМОСТОЯТЕЛЬНОЙ
РАБОТЫ ОБУЧАЮЩИХСЯ**
по дисциплине «Операционные системы, сети, компьютерная графика»
Направление подготовки 01.03.04 Прикладная математика
профиль «Математические методы в экономике»
Форма подготовки очная

Владивосток
2015

План-график выполнения самостоятельной работы по дисциплине

№ п/п	Дата/сроки выполнения	Вид самостоятельной работы	Примерные нормы времени на выполнение	Форма контроля
1	Модуль 1. 4 неделя	Повторение теоретического и практического материала дисциплины, заслушиваемого и конспектируемого в ходе аудиторных занятий; изучение основной и дополнительной литературы, указанной в рабочей учебной программе дисциплины, самоконтроль ответов на основные проблемные вопросы по темам лекций; самостоятельный разбор заданий и задач, решаемых на практических занятиях; самостоятельный повтор действий, осуществляемых в ходе выполнения практических заданий, в том числе при работе со специальным программным обеспечением	8 часов	Собеседование, вопросы 1-2; защита проекта, темы 1-3
2	Модуль 1. 8 неделя	Повторение теоретического и практического материала дисциплины, заслушиваемого и конспектируемого в ходе аудиторных занятий; изучение основной и дополнительной литературы, указанной в рабочей учебной программе дисциплины, самоконтроль ответов на основные проблемные вопросы по темам лекций; самостоятельный разбор заданий и задач, решаемых на практических занятиях; самостоятельный повтор действий, осуществляемых в ходе выполнения практических заданий, в том числе при работе со	8 часов	Собеседование, вопросы 1-5, защита проекта, темы 3-5

		специальным программным обеспечением		
3	Модуль 1. 18 неделя	Повторение теоретического и практического материала дисциплины, заслушиваемого и конспектируемого в ходе аудиторных занятий; изучение основной и дополнительной литературы, указанной в рабочей учебной программе дисциплины, самоконтроль ответов на основные проблемные вопросы по темам лекций; самостоятельный разбор заданий и задач, решаемых на практических занятиях; самостоятельный повтор действий, осуществляемых в ходе выполнения практических заданий, в том числе при работе со специальным программным обеспечением	20 часов	Собеседование, вопросы 16-23; защита проекта, темы 6-7
4	Модуль 2. 4 неделя	Повторение теоретического и практического материала дисциплины, заслушиваемого и конспектируемого в ходе аудиторных занятий; изучение основной и дополнительной литературы, указанной в рабочей учебной программе дисциплины, самоконтроль ответов на основные проблемные вопросы по темам лекций; самостоятельный повтор действий, осуществляемых в ходе выполнения лабораторных работ, в том числе при работе со специальным программным обеспечением	8 часов	Собеседование, вопросы 6-10; защита проекта, тема 7
5	Модуль 2. 8 неделя	Повторение теоретического и практического материала дисциплины, заслушиваемого и конспектируемого в ходе аудиторных занятий; изучение основной и дополнительной	8 часов	Собеседование, вопросы 11-15; защита проекта, тема 8

		литературы, указанной в рабочей учебной программе дисциплины, самоконтроль ответов на основные проблемные вопросы по темам лекций; самостоятельный повтор действий, осуществляемых в ходе выполнения лабораторных работ, в том числе при работе со специальным программным обеспечением		
6	Модуль 2. 12 неделя	Повторение теоретического и практического материала дисциплины, заслушиваемого и конспектируемого в ходе аудиторных занятий; изучение основной и дополнительной литературы, указанной в рабочей учебной программе дисциплины, самоконтроль ответов на основные проблемные вопросы по темам лекций; самостоятельный повтор действий, осуществляемых в ходе выполнения лабораторных работ, в том числе при работе со специальным программным обеспечением	8 часов	Собеседование, вопросы 24-26; защита проекта, темы 9-13
7	Модуль 2. 18 неделя	Повторение теоретического и практического материала дисциплины, заслушиваемого и конспектируемого в ходе аудиторных занятий; изучение основной и дополнительной литературы, указанной в рабочей учебной программе дисциплины, самоконтроль ответов на основные проблемные вопросы по темам лекций	12 часов	Собеседование, вопросы 27-28; защита проектов, темы 14-17
8	Модуль 2. Сессия	Подготовка к экзамену	27 часов	Экзамен

Характеристика заданий для самостоятельной работы обучающихся и методические рекомендации по их выполнению

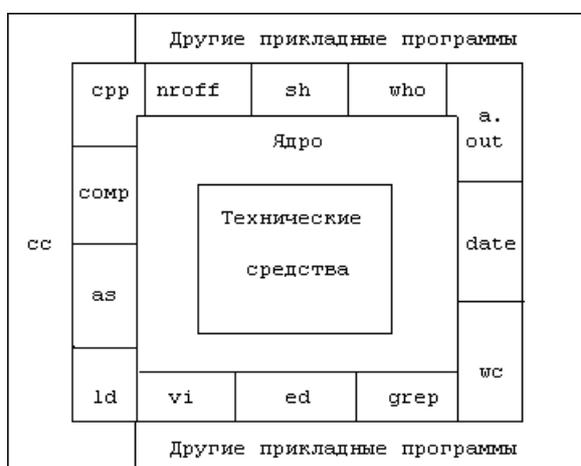
Задания и методические рекомендации по теме «Операционные системы»

Необходимо изучить следующий материал и ответить на вопросы по данному материалу.

Архитектура Unix и Windows

Источник: Архитектура UNIX и Windows - Статьи Software (статья). URL: <http://fcenter.ru/online/softarticles/os/6773>

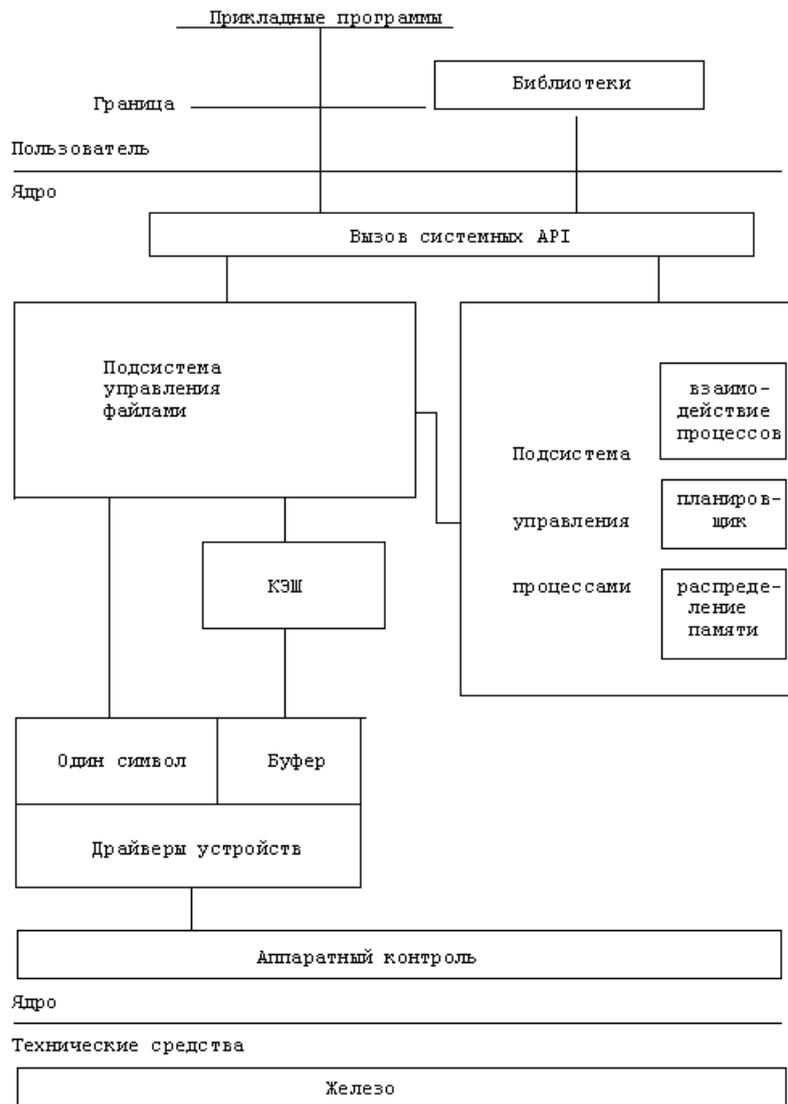
Структуру UNIX проще всего представить в виде двух слоев. Первым является ядро. Оно непосредственно взаимодействует с железом и обеспечивает переносимость всего остального ПО на компьютеры с разным аппаратным обеспечением. Ядро предоставляет программам определенный набор системных API, с помощью которых производится создание процессов, управление ими, их взаимодействие и синхронизация, а также файловый ввод/вывод. Вторым слоем является программное обеспечение, прикладное или системное: командный интерпретатор, графическая оболочка и т. д.



Структура ОС UNIX

Заглянем глубже в ядро системы. Оно позволяет всем остальным программам общаться с периферийными устройствами, регулирует доступ к файлам, управляет память и процессами. Ядро - это связной, к которому обращаются посредством системных вызовов (запрашивая какую-то услугу).

Связь эта - не односторонняя: ядро может и возвращать в случае необходимости какие-то данные. Основным достоинством ядра является строгая стандартизация системных API. За счет этого во многом достигается переносимость кода между разными версиями UNIX и абсолютно различным аппаратным обеспечением.



Структура ядра UNIX

Все обращения к ядру системы можно разделить на две категории: программа вызывает подсистему управления файлами или подсистему управления процессами. Первая отвечает за все, что связано с файлами: управление, размещение, доступ. Процессы же – это, в общем случае, любые

запущенные программы. Поэтому подсистема управления процессами служит для их жизнеспособности, синхронизации и управления. Важно так же и то, что файловая подсистема и подсистема управления процессами могут общаться друг с другом: любой процесс может вызывать системные API для работы с файлами. Прелесть UNIX состоит в том, что эти API универсальны (да и в Windows наблюдается та же картина). Вот самые главные из них: open, close, read, write, stat, chown, chmod (суть почти всех вызовов интуитивно понятна из названия, кроме, разве что, последних трех, поэтому поясню - они служат для управления атрибутами файлов, информации о владельце и прав доступа) и др. Каждый из этих системных вызовов в программе на языке C является обычной функцией. Информацию по любому из них можно запросто найти в man.

Подсистема управления файлами – почти единственная из всех работает с драйверами, которые являются модулями ядра. "Почти", потому что есть еще и сетевая подсистема, которая работает, например, с драйвером сетевой карты и с драйверами различных современных сетевых устройств. Ее, однако, мы рассматривать не будем. Обмен данными с драйверами может проходить двумя способами: с помощью буфера или потока. Суть первого метода заключается в том, что для информации выделяется кэш (или сверхоперативная память, как его называли раньше), в который заносится необходимый блок данных. Далее информация из кэша передается к драйверу. Драйвер - единственный элемент ядра, способный управлять периферийными устройствами. Но подсистема управления файлами может взаимодействовать с драйвером и через поток. Поток представляет собой посимвольную передачу данных драйверу. Следует отметить, что способ взаимодействия с драйвером определяется не пользователем и не приложением. Он является характеристикой того устройства, которым управляет драйвер. Очевидно, что потоковое общение позволяет взаимодействовать более оперативно, чем общение через буфер. Ведь на заполнение буфера тратится время и, следовательно, возрастает время отклика.

Теперь более подробно рассмотрим подсистему управления процессами. Она отвечает за синхронизацию и взаимодействие процессов, распределение памяти и планирование выполнения процессов. Для всех этих целей в подсистему управления процессами включены три модуля, которые наглядно продемонстрированы на схеме. Хорошим примером взаимодействия подсистем управления файлами и процессами является загрузка файла на исполнение. В этом случае подсистеме управления процессами требуется обратиться к коллеге, чтобы считать исполняемые файлы.

Чуть выше мы перечисляли системные API для управления файлами. Теперь рассмотрим вызовы, служащие для работы с процессами: `fork` (создает новый процесс), `exec` (выполняет процесс), `exit` (завершает исполнение процесса), `wait` (один из способов синхронизации), `brk` (управляет памятью, выделенной процессу), `signal` (обработчики исключений) и др.

Следующие два модуля являются очень важными в понимании всей подсистемы управления процессами. Первый - модуль распределения памяти, позволяет избежать нехватки оперативной памяти. Хотя механизм свопинга и файлов подкачки (технически правильно это, кстати, называется виртуальной памятью) уже ни для кого не секрет, в тени остается другой факт: операционная система (в лице описываемой подсистемы) может либо скидывать все данные, относящиеся к конкретному процессу, на диск, либо скидывать страницы памяти (страничное замещение). Таким образом, модуль распределения памяти выполняет очень важную функцию - он определяет какому процессу сколько выделить памяти.

Виртуальная память была изобретена в 1962 году, в Англии при создании суперкомпьютера Atlas. В большинстве современных компьютеров оперативная память не так велика, как используемое процессором адресное пространство. Размер ОЗУ типичного персонального компьютера

варьируется от десятков до сотен мегабайт. При запуске программа загружается с какого-либо накопителя в оперативную память. Если же программа не помещается в ОЗУ, то те её части, которые в данный момент не выполняются, хранятся во вторичном запоминающем устройстве, чаще всего винчестере, и такая память называется виртуальной. Безусловно, перед выполнением необходимая часть программы должна быть перемещена в оперативную память. Данные функции выполняет ядро операционной системы (диспетчер виртуальной памяти, находящийся в микроядре). И для программы и для пользователя эти действия прозрачны. Естественно, на запросы к виртуальной памяти уходит гораздо большее время, нежели к ОЗУ.

Второй модуль – планировщик. Его задача не менее важна. UNIX – мультизадачная ОС, то есть одновременно может выполняться множество процессов. Мы, однако, знаем, что в фиксированный момент времени на одном процессоре может выполняться только одна команда. Именно поэтому нужен виртуальный рефери, который будет определять, какому процессу исполняться сейчас, а какому - через секунду. На практике же планировщик переключает контекст, то есть перед тем, как остановить исполнение какого-то процесса, он запоминает состояние регистров, памяти и т. д., а уже после этого запускает другой процесс в его собственном адресном пространстве. И еще один тонкий момент: каждый запущенный процесс "думает", что он единственный. Дополнительно существует механизм приоритетов. Очевидно, чем выше приоритет, тем быстрее начнет исполняться процесс. Процессы могут также обмениваться между собой информацией. В случае их синхронного взаимодействия синхронизацию осуществляет модуль взаимодействия (например, функция wait).

Вот мы и подошли к последнему уровню – аппаратному контролю. На данном уровне происходит обработка прерываний и связь ядра с железом. Здесь следует отметить лишь пару моментов, во-первых, прерывания могут

"прерывать" работу процессора и требовать внимания к себе (после этого процессор без проблем возвращается к выполнению оставленных процессов), а, во-вторых, обработку прерываний осуществляют специальные функции ядра.

Windows 2000/XP построены на архитектуре микроядра (microkernel architecture). ОС Windows 95/98 используют монолитное (monolithic) ядро. Микроядра являются сравнительно небольшими и модульными. Благодаря последнему новые устройства зачастую добавляются как модули, которые можно загружать/выгружать на этапе исполнения без перекомпиляции ядра. На архитектуре микроядра построены также FreeBSD и Mac OS X. Монолитные же ядра используются еще и в Linux. Они оптимизированы для более высокой производительности с минимальными контекстными переключениями. Такая архитектура упрощает поддержку кода ядра для разработчиков, но требует перекомпиляции ядра при добавлении новых устройств. Следует отметить, что описанные здесь различия являются "классическими", на практике монолитные ядра могут поддерживать модульность (что зачастую и происходит), а микроядра могут требовать перекомпиляции.

Архитектура Windows

Структура Windows 2000/XP не отличается оригинальностью: ядро системы (исполняется на максимально приоритетном уровне процессора) и пользовательские подсистемы (исполняются на минимально приоритетном уровне).

Ядро системы является критичной частью кода, любые ошибки, происходящие в ядре, приводят к фатальному краху системы – "синему экрану". Фактически – это ошибки типа "Нарушение общей защиты". Как только код ядра начинает обращаться в запрещенные для него области памяти (попытка прочитать или записать данные, исполнить неверную инструкцию, переход на запрещенную область), срабатывает система защиты памяти

процессора, и управление передается системному обработчику исключений. Обработчик исключений не может восстановить корректное поведение кода. Все, что он делает – это вывод дампа на синий экран с указанием типа ошибки и содержимого памяти в области, где сработала защита. Пользовательские подсистемы не столь критично влияют на работу системы в целом, так как они изолированы друг от друга и от ядра средствами управления памятью и собственно процессором. Ошибки, возникающие в приложениях, исполняются на уровне пользователя, то есть на менее привилегированном уровне, нежели ядро. Поэтому, система в состоянии контролировать процесс. При возникновении же ошибки или сбоя управление передается обработчику ошибок, который называется "Doctor Watson". Он принудительно завершит приложение. Ядро системы и остальные подсистемы остаются в целостности и сохранности.

Ядро UNIX/Linux имеет два вида исключений, которые обычно называют "oops" и "panic". Почти в каждой операционной системе паника происходит в тех случаях, когда ядро обнаруживает серьезную неисправность. Если система каким-либо образом повредила сама себя, ей требуется остановиться немедленно, пока она не произведет необратимых критических изменений (типа уничтожения файловой системы). Везде, где только возможно, UNIX/Linux пытается детектировать проблему и справиться с ней без остановки всей системы. Например, многие ситуации типа "oops" приводят к завершению процесса, который нормально запустился, но потом зациклил систему. Бывают, однако, ситуации, когда все настолько плохо, что полная паника является наилучшим выходом. Считается, что пользователи стабильных версий ядра не должны встречать ни "паник", ни "oops". Но в реальном мире они иногда происходят.

Недавно найденный "TF-баг" (смотрите [здесь](#)) является хорошим примером паники. Процессор пытается передать управление процессу,

которого не существует. Это приводит к краху всей системы. В данном случае, у системы нет другой альтернативы, чем запаниковать.

Ядро, поставляемое с Red Hat Linux 7.3 (и некоторыми другими дистрибутивами), содержит баг в файловой системе ext3. Эта ошибка приводит к "oops", завершая время от времени некоторые процессы (также этот баг приводит к замедлению всей системы). Хотя данная ошибка уже исправлена (патч есть и в обновлении от Red Hat), этот случай познакомил многих пользователей с ошибками типа "oops".

Ядро Windows 2000/XP состоит из нескольких системных компонентов, каждый из которых отвечает за определенный набор задач. Основные компоненты ядра:

Микроядро (Microkernel) - компактный код, можно сказать, сердце системы. В рамках микроядра работают ключевые службы: диспетчер памяти, диспетчер задач и другие.

Слой абстрагирования (Hardware Abstraction Layer, HAL). Полностью абстрагирует код системы от конкретного аппаратного оборудования. Использование HAL позволяет обеспечить переносимость 99% кода системы между различным оборудованием.

Диспетчер Ввода/Вывода (Input/Output Manager). Полностью контролирует потоки обмена между системой и устройствами. Драйверы устройств работают в контексте I/O Manager. Если драйвер написан с ошибками и может привести к сбою - это вызовет фатальный крах ядра и всей системы. 70% случаев фатальных сбоев ("синий экран") - есть результат некорректного поведения драйверов устройств.

Windows XP содержит встроенный механизм контроля драйверов: правильно

написанный и тщательно протестированный драйвер поставляется с цифровой подписью (Driver Signing). Правильная настройка системы заключается в запрещении установки драйверов без корректной подписи.

Модуль управления объектами (Object Manager), управления виртуальной памятью (Virtual Memory Manager), управления процессами (Process Manager), управления безопасностью (Security Reference Monitor), управления локальными вызовами (Local Procedure Calls Facilities) - важные компоненты ядра системы подробно рассматриваться не будут. Наконец, особое по значению и важности место в ядре системы занимает модуль графического интерфейса - Win32k.sys. Фактически - это часть подсистемы Win32, отвечающая за прорисовку и управление графическим интерфейсом. Этот модуль расположен в ядре специально для того, чтобы существенно повысить производительность графических операций ввода/вывода. Однако размещение столь критической части в ядре накладывает чрезвычайно строгие требования к корректности его исполнения. Фактически, ошибка в коде Win32k.sys приведет к краху системы. Разработчики Windows уделяют огромное внимание этому модулю, и именно он наиболее тщательно протестирован. Опыт эксплуатации систем Windows показывает, что код Win32k.sys работает абсолютно корректно и не содержит фатальных ошибок. Однако некорректный драйвер видеосистемы может все испортить.

В Windows также реализованы дополнительные функции для повышения стабильности работы ОС. Система защиты файлов Windows автоматически предотвращает случайные изменения системных файлов операционной системы, вносимые пользователем или приложениями, эффективно защищая всю систему в целом. То есть, когда какая-то программа внесла изменения или просто заменила системные файлы Windows, считая, что у программы более новые, Windows отслеживает изменения и уведомляет

об этом пользователя, говоря, что для стабильности системы желательно восстановить исходные файлы. Так же существует поддержка нескольких версий DLL, что повышает совместимость приложений и повышает стабильность.

Итог

Различия между Windows и UNIX прокомментировали разработчики из компании ASPLinux.

"Операционные системы Unix и Windows достаточно сильно отличаются в реализации различных сервисов и служб. В соответствии с темами, затронутыми в этой статье, можно отметить несколько глобальных различий.

В Unix/Linux графическая система существует отдельно от ядра и функционирует как обычное приложение. В операционных системах Windows графическая система интегрирована в ядро. В случае использования операционной системы на рабочей станции, особенно при запуске графичеомких приложений, возможно, лучше, когда графическая система входит в ядро - в этом случае она может быстрее работать. А при работе на сервере предпочтительней отделение графической системы от ядра ОС, так как она загружает память и процессор. В случае Unix/Linux графическую систему можно просто отключить, к тому же, если системный администратор ее все-таки хочет использовать, в Linux есть несколько графических оболочек на выбор, некоторые из них (например, WindowMaker) достаточно слабо загружают машину. Эта же особенность Unix-образных операционных систем позволяет запускать эти ОС на машинах с весьма скромными объемами ОЗУ и т.п. В случае Windows же графическая система слишком тесно интегрирована в ОС, поэтому она должна запускаться даже на тех серверах, на которых она вовсе не нужна.

Отметим также методика разделения прав доступа в *Windows 2000* и *Unix/Linux*. В первом - разделение прав доступа основано на *ACL (access control lists)*, то есть, к примеру, можно настроить систему таким образом, чтобы администратор не имел возможности управлять файлами пользователей. У *Unix/Linux* же всегда есть суперпользователь - *root*, который имеет доступ абсолютно ко всему. То есть теоретически модель безопасности в *Windows* лучше: чтобы полностью завладеть хорошо настроенной системой *Windows*, хакеру придется ломать больше, в *Unix/Linux* же достаточно взломать доступ к *root*. (В *Unix/Linux* используются более старые технологии, тем не менее, некоторые дистрибутивы *Linux* сейчас начинают поддерживать *ACL*, среди них - *ASPLinux 7.3 Server Edition*). Но теория несколько смазывается практикой с той стороны, что в *Windows* не так быстро, как в *Linux*, заделываются "дыры", что уже относится к плюсам открытой модели разработки. В результате оказывается, что в *Windows* по статистике больше дыр, через которые злоумышленник может пробраться в систему. Но, опять же, точно о количестве дыр в *Linux* и *Windows* можно будет сказать только тогда, когда количество пользователей обоих видов ОС будет примерно одинаковым.

В *Linux* поддерживаются несколько файловых систем, наиболее продвинутые - это *Ext2*, *Ext3*, *XFS*. ОС *Windows* завязана по большому счету на одну файловую систему - *NTFS* или *FAT 32*. Файловые системы *Ext2*, *Ext3*, *XFS* по оценкам работают быстрее. Принципиальное же отличие в том, что в *UNIX/Linux* вообще нет понятия диска, физического или логического. Вся работа с устройствами хранения данных организуется через специальные файлы устройств, которые отображают физический носитель (диск, лента и т. п.) или его части (разделы) в файловую систему.

Важное отличие - наличие в Windows технологии ActiveX, нечто подобное в Unix/Linux реализуется с помощью CORBA и Wopobo. Эта технология, с одной стороны, предоставляет пользователю множество удобств, с другой стороны - она же допускала в свое время такие вещи, как автоматический запуск Outlook'ом вируса, пришедшего по почте. Одно из важных отличий этих технологий в том, что элементы ActiveX могут внедряться в текст HTML, что имеет как ряд достоинств, так и недостатков.

Можно перечислить еще ряд отличий Unix-подобных операционных систем от Windows, например, встроенную поддержку удаленного доступа в Unix и отсутствие оной в Windows по умолчанию (она реализуется в серверных версиях Windows, а также с помощью дополнительных средств, например, Citrix). В Unix/Linux и Windows сильно различаются сетевые подсистемы (IP-stack), по ряду оценок сетевая подсистема

<i>Unix/Linux</i>	<i>эффективнее.</i>
-------------------	---------------------

Можно было бы упомянуть богатый набор ПО, которое может поставляться вместе с Linux, между тем, Windows также развивается в этом направлении. Дополнительные отличия же в архитектуре в основном сводятся к отличиям работы монолитных и модульных ядер, которые также зачастую не являются преимуществами или недостатками, а просто отличиями. При всем при этом можно с уверенностью сказать, что характеристики работы Windows или Linux гораздо больше зависят от аккуратности и квалификации пользователя, чем от архитектуры той или иной ОС".

Задания и методические рекомендации по теме «Язык программирования Python»

Источник: Маккинли У. Python и анализ данных. — М.: ДМК Пресс, 2015.

Интерпретатор Python

Python – *интерпретируемый* язык. Интерпретатор Python исполняет программу по одному предложению за раз. Стандартный интерактивный интерпретатор Python запускается из командной строки командой `python`:

```
$ python
Python 2.7.2 (default, Oct 4 2011, 20:06:09)
[GCC 4.6.1] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> a = 5
>>> print a
5
```

Строка `>>>` – это приглашение к вводу выражения. Для выхода из интерпретатора Python и возврата в командную строку нужно либо ввести команду `exit()`, либо нажать `Ctrl-D`.

Для выполнения Python-программы нужно просто набрать команду `python`, указав в качестве первого аргумента имя файла с расширением `.py`. Допустим, вы создали файл `hello_world.py` с таким содержимым:

```
print 'Hello world'
```

Чтобы выполнить его, достаточно ввести следующую команду:

```
$ python hello_world.py
Hello world
```

Многие программисты выполняют свой код на Python именно таким образом, но в мире *научных* приложений принято использовать IPython, улучшенный и дополненный интерпретатор Python. Ему посвящена вся глава 3. С помощью команды `%run` IPython исполняет код в указанном файле в том же процессе, что позволяет интерактивно изучать результаты по завершении выполнения.

```
$ ipython
Python 2.7.2 |EPD 7.1-2 (64-bit)| (default, Jul 3 2011, 15:17:51)
Type "copyright", "credits" or "license" for more information.
```

```
IPython 0.12 -- An enhanced Interactive Python.
?      -> Introduction and overview of IPython's features.
%quickref -> Quick reference.
help    -> Python's own help system.
object? -> Details about 'object', use 'object??' for extra details.

In [1]: %run hello_world.py
```

Основы



```
Hello world
```

```
In [2]:
```

По умолчанию приглашение IPython содержит не стандартную строку `>>>`, а строку вида `In [2]:`, включающую порядковый номер предложения.

Источник: Онлайн-курс на Курсере «Математика и Python для анализа данных». МФТИ, Академия Яндекса.
<https://www.coursera.org/learn/mathematics-and-python>

Задание 1

1. Перейдите на сайт [continuum.io](https://www.continuum.io) по ссылке <https://www.continuum.io/downloads>
2. Выберите подходящую Вам операционную систему и перейдите в соответствующий раздел сайта.
3. Следуя инструкциям на сайте, установите Python 3.5.

Задание 2

1. Запустите `ipython notebook`. Команда запуска может несколько отличаться от команды запуска в инструкции из видео. Например, команда может быть такой: `ipython-3.5 notebook`
2. Создайте новый файл типа `.ipynb`
3. Создайте новую ячейку. В ней импортируйте библиотеку `numpy` (`import numpy`) и выведите на экран версию библиотеки (`numpy.__version__`)
4. Создайте новую ячейку. В ней импортируйте библиотеку `scipy` (`import scipy`) и выведите на экран версию библиотеки (`scipy.__version__`)
5. Создайте новую ячейку. В ней импортируйте библиотеку `pandas` (`import pandas`) и выведите на экран версию библиотеки (`pandas.__version__`)
6. Создайте новую ячейку. В ней импортируйте библиотеку `matplotlib` (`import matplotlib`) и выведите на экран версию библиотеки (`matplotlib.__version__`)
7. Сделайте скриншот №1, на котором будет хорошо видно результаты вашей работы, и загрузите его в форму.

Задание 3

1. Запустите `ipython notebook`. Команда запуска может несколько отличаться от команды запуска в инструкции, например, команда может быть такой `ipython-2.7 notebook`
2. Создайте новый файл типа `.ipynb`
3. В файле создайте новую ячейку и измените её тип на `Markdown`
4. В первой строке созданной ячейки наберите название специализации “Машинное обучение и анализ данных” и сделайте эту строку заголовком уровня 1.
5. В следующей строке созданной ячейки наберите название нашего курса “Математика и Python” и сделайте строку заголовком уровня 2.
6. В третьей строке ячейки наберите текст "Задание 1".
7. Запустите выполнение ячейки.
8. Сделайте скриншот №2, на котором будет хорошо видно результаты вашей работы, и загрузите его в форму.

В результате работы вы установите на компьютер Python и библиотеки, необходимые для дальнейшего прохождения курса.

Вы также научитесь запускать `iPython notebook` и выполнять в нем простые команды.

Выполнение задания будет проверяться на основе двух скриншотов, которые вы прикрепите к заданию.

Программирование на Python

1. Типы данных в Python

Настало время познакомиться с типами данных в Python. Числовые типы `int` и `float` уже были упомянуты ранее:

```
x = 3.5
print type(x) % OUT: <type 'float'>

y = 4
print type(y) % OUT: <type 'int'>
```

С помощью функции `type` можно узнать код произвольного объекта. Строки, например, могут быть двух типов:

```
print type("abs") % OUT: <type 'str'>
print type(u"abs") % OUT: <type 'unicode'>
```

Отличия между ними будут подробнее обсуждаться позднее.

Упорядоченный набор значений в Python можно представить с помощью списка:

```
w = [1, 2, 3]
print type(w) % OUT: <type 'list'>
```

Списки в Python очень похожи на знакомые по другим языкам программирования массивы, но, в отличие от массивов, могут содержать элементы разных типов.

Python является языком с утиной типизацией. В языках с динамической типизацией, частным случаем которой является утиная типизация, тип переменной определяется не в момент её объявления, а в момент присваивания значения. Следовательно, в разных участках кода переменная может принимать значения разных типов. В случае утиной типизации границы применимости объекта определяются не столько конкретным типом и иерархией наследования, сколько наличием и отсутствием у него определенных методов и свойств. Термин происходит от шуточного «утинового теста»: «Если нечто выглядит, плавает и крикает как утка, то это, вероятно, и есть утка».

Другой способ представить упорядоченный набор значений состоит в определении кортежа (`tuple`). В Python кортежи записываются в круглых скобках:

```
print type((1, 2, 3)) % OUT: <type 'tuple'>
```

Кортеж относится к так называемым неизменяемым типам данных, чем существенно отличается от списка. Попытка изменить кортеж или добавить в него еще один элемент приведет к возникновению ошибки. Эта особенность позволяет использовать кортежи в качестве ключей словаря.

Неупорядоченный набор значений представим в виде множеств, которые записываются в фигурных скобках:

```
print type({1, 2, 3}) % OUT: <type 'set'>
```

Еще один тип данных, словарь, позволяет хранить в себе пары вида ключ-значение. Создать словарь можно с помощью конструктора `dict()`:

```
e = dict()      % Создан пустой словарь
e['abc'] = 3.5  % Строке-ключу 'abc' поставлено в
                % соответствии значение 3.5 типа float
```

Ключами в словаре могут быть не только строки. В качестве ключа допустимо использовать, например, числовые значения и кортежи:

```
e[3.5]         = 'abc'
e[(1, 2, 3)]   = 4.5
```

1

Но использование списка в качестве ключа недопустимо, так как список является изменяемым.

Синтаксис для чтения данных из словаря по ключу аналогичен синтаксису для получения данных по индексу списка. Грубо говоря, ключ есть обобщение понятия индекса списка: ключ может принимать значение произвольного хэшируемого типа. К слову, список не является хэшируемым, и именно поэтому не может быть выбран в качестве ключа.

В Python словарь использует хэш-таблицы. Идея заключается в том, чтобы заменить сравнение ключей более быстрым сравнением их хэш-значений. В таком случае важно, чтобы все ключи были хэшируемы, то есть хэш-значение не менялось во время исполнения программы, а равные ключи-объекты имели одинаковое хэш-значение.

Ранее было сказано, что множества являются изменяемыми объектами, а значит не могут выступать в роли ключей. Но в качестве ключей можно использовать неизменяемые множества, которые можно получить используя функцию `frozenset()`:

```
s = frozenset(1,2,3)
e[s] = 3.76
```

Таким образом, к данному моменту были изучены основные типы данных в Python и подробно было рассказано о существенных отличиях изменяемых и неизменяемых типов. На следующем занятии будут рассмотрены основные управляющие конструкции языка Python.

2. Синтаксис языка Python

В Python оператор условного перехода выглядит привычным образом. Например, пусть дан следующий код:

```
if x:
    print "OK"
else:
    print "NOT OK"
```

Если переменная `x` была равна `True`, то исполнится код в теле инструкции `if` и на экран будет выведено ОК. В противном случае, например если `x` определена как `False`, исполнится код в теле инструкции `else`. В Python для выделения блоков кода используются отступы: отступы инструкций в пределах одного блока должны совпадать по величине.

Другой пример. Следующий код выводит числа от 0 до 9:

```
for i in range(10):
    print i
```

При этом в `range()` можно указать начальное значение `i`. Например, вот этот код выводит числа от 1 до 9.

```
for i in range(1,10):
```

```
print i
```

В данных примерах каждое число выводится на отдельной строке, поскольку `print` по умолчанию начинает новую строчку после вывода требуемого значения. Чтобы все числа выводились в одной строке, необходимо добавить запятую в конец строчки с `print` следующим образом:

```
for i in range(1,10):
    print i,
```

Теперь числа от 1 до 9 выводятся в одной строке через пробел.

На самом деле `range()` это функция, которая возвращает список натуральных чисел в определяемом её аргументами диапазоне:

```
print range(2,5)
OUT: [2,3,4]
```

Использовать `range()` совершенно не обязательно. Циклы в Python перебирают по очереди элементы списка, стоящего после `in`. Например, можно написать так:

```
for i in [2,3,4]:
    print i,
```

2

МФТИ

Специализация «Машинное обучение и анализ данных»

В Python 2 кроме функции `range()` существует и функция `xrange()`, которую также можно использовать в цикле:

```
for i in xrange(2,5):
    print i,
OUT: 2, 3, 4
```

Использовать `range()` совершенно не обязательно. Циклы в Python перебирают по очереди элементы списка, стоящего после `in`. Например, можно написать так:

```
for i in [2,3,4]:
    print i,
```

2

МФТИ

Специализация «Машинное обучение и анализ данных»

В Python 2 кроме функции `range()` существует и функция `xrange()`, которую также можно использовать в цикле:

```
for i in xrange(2,5):
    print i,
OUT: 2, 3, 4
```

Результат выполнения не отличается от такового при использовании `range()`. Но можно убедиться, что `range` и `xrange` возвращают объекты разных типов:

```
print type(range(2,5))
OUT: list
print type(xrange(2,5))
OUT: iter
```

Функция `xrange` возвращает не список, а так называемый генератор. В то время, как при использовании `range()` сначала создается список, а только потом происходит итерирование по этому списку, при использовании генераторов список никогда не создается и не занимает память. Это особенно актуально, когда необходимо итерировать в очень большом диапазоне значений.

```
print xrange(2,500000000)
OUT: xrange(2,500000000)
```

В Python существует способ удобно и компактно задавать некоторые списки с помощью так называемого конструктора списка (англ. list comprehension.):

```
w = [ x ** 2 for x in range(1,11) ]
print w
OUT: [ 1, 4, 9, 16, 25, 36, 49, 64, 81, 100 ]
print type(w)
OUT: <type 'list'>
```

В этом примере создан список квадратов чисел от 1 до 10. Если необходимо построить список, например, квадратов только четных чисел из этого диапазона, конструктор допускает использование условия:

```
w = [ x ** 2 for x in range(1,11) if x % 2 == 0 ]
OUT: [ 4, 16, 36, 64, 100 ]
```

Аналогично выглядит конструктор генераторов:

```
w = ( x ** 2 for x in range(1,11) if x % 2 == 0 )
OUT: ( 4, 16, 36, 64, 100 )
print type(w)
OUT: <type 'generator'>
```

В этом случае весь список не будет храниться в памяти во время работы цикла.

Также в Python доступен цикл `while`, который продолжает выполнение, пока выполнено указанное условие, а также операторы `break`, который досрочно прерывает цикл, и `continue`, который начинает следующий проход цикла, минуя оставшееся тело цикла. Например:

```
s = 0
while True: % Это условие всегда выполняется
    s += 1
    if s % 2 ==0: % Через раз
        print "Continue" % вместо s будет выведено "continue"
        continue % и код ниже в таком случае не будет исполнен в этой итерации
    print s
    if s > 10: % С помощью этой конструкции
        break % <<бесконечный цикл досрочно прерывается>>
```

Важно уметь определять свои функции. Например, следующий код представляет собой альтернативную реализацию уже известной функции `range`:

3

```
def myrange(a,b):
    res = [ ]
    s = a
    while s!=b:
        res.append(s)
        s+=1
    return res
```

Этот пример демонстрирует синтаксис для создания функций. На самом деле при создании функции необходимо обработать особые случаи, но это не является текущей целью.

Функции в Python являются объектами первого класса: их можно передавать в качестве аргументов, присваивать их переменным и так далее. Например, функция `map()` позволяет обрабатывать одну или несколько последовательностей с помощью переданной в качестве аргумента функции:

```
def sq(x):
    return x ** 2

print map(sq, range(10))
OUT: [ 1, 4, 9, 16, 25, 36, 49, 64, 81]
```

Функция в Python может быть определена как с помощью оператора `def`, так и с помощью лямбда-выражения. Следующие операторы эквивалентны:

```
def sq(x):
    return x ** 2

sq = lambda x: x**2
```

Задания и методические рекомендации по теме «Сети»

1. Изучить рекомендуемую литературу по данной теме.
2. Реализовать клиент-серверное приложение с использованием API GoogleMaps, позволяющее пользователю искать географические данные по введенному названию. Для этого воспользоваться сетевой библиотекой языка Python `urllib`.

Задания и методические рекомендации по теме «Компьютерная графика»

1. Изучить рекомендуемую литературу по данной теме.
2. Изучить библиотеку `matplotlib` для графической визуализации данных.
3. Применить методы из библиотеки `matplotlib` для обработки и анализа данных.

4. Ознакомиться с другими средствами визуализации (см. ниже).

Источник: Маккинли У. Python и анализ данных. — М.: ДМК Пресс, 2015.

Инструментальная экосистема визуализации для Python

Как обычно бывает в проектах с открытым исходным кодом, в средствах создания графики для Python нехватки не ощущается (их слишком много, чтобы все перечислить). Помимо открытого кода, существуют также многочисленные коммерческие библиотеки с интерфейсом к Python.

В этой главе и в книге в целом я использую, в основном пакет `matplotlib`, потому что это наиболее популярное средство построения графиков в Python. Но хотя `matplotlib` является важной частью научной экосистемы Python, у него обнаруживается немало недостатков, когда дело доходит до создания и отображения статистических графиков. Пользователям MATLAB пакет `matplotlib` покажется знакомым, тогда как пользователи R (особенно работающие с великолепными пакетами `ggplot2` и `trellis`), наверное, испытают разочарование (по крайней мере, на момент написания этой книги). С помощью `matplotlib` можно создать отличные графики для отображения в веб, но для этого обычно приходится прикладывать немало усилий, потому что библиотека проектировалась для полиграфических изданий. Но если не обращать внимания на эстетические нюансы, то для большинства задач этого достаточно. В `pandas` я, наряду с другими разработчиками, стремился предложить пользователям удобный интерфейс, позволяющий упростить построение большинства графиков, применяемых в анализе данных.

Но существуют и активно используются и другие средства визуализации. Ниже я перечислю некоторых из них и призываю вас самостоятельно исследовать экосистему.

Chaco

Пакет `Chaco` (<http://code.enthought.com/chaco/>), разработанный компанией `Enthought`, представляет собой инструментарий для построения графиков. Он

годится как для рисования статических графиков, так и для интерактивной визуализации. Особенно хорош он, когда нужно выразить сложные визуализации, характеризуемые наличием взаимозависимостей между данными. По сравнению с `matplotlib` `Chaco` гораздо лучше поддерживает взаимодействие с элементами графика, а отрисовка производится очень быстро, так что этот пакет будет хорошим выбором для построения интерактивных приложений с графическим интерфейсом.

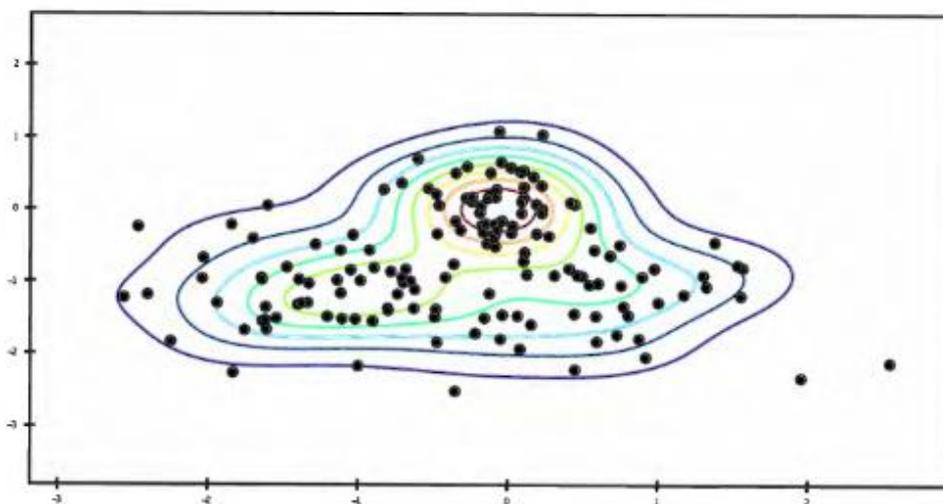


Рис. 8.26. Пример графика, построенного `Chaco`

mayavi

Проект `mayavi`, разработанный Прабху Рамачандраном (Prabhu Ramachandran) Гаэлем Вароко (Gaël Varoquaux) и другими, – это библиотека трехмерной графики, построенная поверх написанной на C++ библиотеки VTK с открытым исходным кодом. `mayavi`, как и `matplotlib`, интегрирована с IPython, поэтому с ней легко работать интерактивно. Графики можно панорамировать, поворачивать и масштабировать с помощью мыши и клавиатуры. Я использовал `mayavi` для создания од-

ной иллюстрации укладывания в главе 12. И хотя я не привожу здесь свой код работы с `matplotlib`, в сети достаточно документации и примеров. Я полагаю, что во многих случаях это неплохая альтернатива таким технологиям, как WebGL, хотя интерактивными графиками труднее поделиться.



Прочие пакеты

Разумеется, для Python хватает и других библиотек и приложений для визуализации: `PyQwt`, `Veusz`, `gnuplot-py`, `biggles` и т. д. Я видел, как удачное применение `PyQwt` в графических приложениях, написанных с помощью каркаса `Qt` и надстройки `PyQt` над ним. Многие библиотеки продолжают активно развиваться (некоторые являются частью гораздо более крупных приложений), но в последние годы наблюдается общая тенденция к дрейфу в сторону веб-технологий и отхода от графики для настольных компьютеров. Я еще скажу об этом несколько слов в следующем разделе.

Будущее средств визуализации

Похоже, будущее за веб-технологиями (на основе JavaScript), и от этого никуда не деться. Без сомнения, за прошедшие годы вы встречались с самыми разными видами статических и интерактивных визуализаций, написанных на Flash или JavaScript. Постоянно появляются все новые и новые инструменты (например, `d3.js` и многочисленные отпочковавшиеся от него проекты). Напротив, разработка средств визуализации на платформах, отличных от веб, в последние годы резко замедлилась. Это относится как к Python, так и к другим средам для анализа данных

Требования к представлению и оформлению результатов самостоятельной работы

Самостоятельная работа включает в себя повторение теоретического и практического материала дисциплины, заслушиваемого и конспектируемого в ходе аудиторных занятий; изучение основной и дополнительной литературы, указанной в рабочей учебной программе дисциплины, самоконтроль ответов на основные проблемные вопросы по темам занятий; самостоятельный повтор действий, осуществляемых в ходе выполнения лабораторных работ и практических занятий, в том числе при работе со специальным программным обеспечением — средой разработки для языка программирования Python.

Результаты самостоятельной работы представляются и оформляются в виде ответов на основные положения теоретического и практического материала дисциплины по темам; работающего исходного кода результата решения практических заданий и задач; собственных действий, осуществляемых в ходе выполнения лабораторных работ.

Критерии оценки выполнения самостоятельной работы

Общие критерии оценки выполнения самостоятельной работы – правильность ответов на вопросы по темам теоретической части дисциплины, верность получаемых ответов в ходе решения практических заданий и задач, достижение правильного результата при осуществлении собственных действий по лабораторным работам.

Оценивание знаний в форме собеседования проводится по критериям:

- логичность изложения, знание и понимание основных теоретических и практических положений и концепций по теме;

- владение теоретическими и практическими приёмами написания, отладки и тестирования программ на языке программирования Python.

Оценивание знаний в форме защиты проекта проводится по критериям:

- завершенность и полнота выполненных заданий в рамках данного проекта;

- владение методами и приемами решения конкретных задач и самостоятельность использования специализированного программного обеспечения;

- качество оформления отчёта в соответствии с правилами и стандартами оформления документации разработанных программ.



МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ
ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение
высшего профессионального образования
«Дальневосточный федеральный университет»
(ДВФУ)

ШКОЛА ЕСТЕСТВЕННЫХ НАУК

ФОНД ОЦЕНОЧНЫХ СРЕДСТВ
по дисциплине «Операционные системы, сети, компьютерная графика»
Направление подготовки 01.03.04 Прикладная математика
профиль «Математические методы в экономике»
Форма подготовки очная

Владивосток
2015

**Паспорт
фонда оценочных средств
по дисциплине «Операционные системы, сети, компьютерная графика»**

Код и формулировка компетенции	Этапы формирования компетенции	
ОПК-2 - способность использовать современные математические методы и современные прикладные программные средства и осваивать современные технологии программирования	Знает	современные технологии программирования на языке Python; современные математические методы машинного обучения
	Умеет	разрабатывать программные продукты с помощью современных технологий программирования
	Владеет	эффективными методами разработки программных продуктов с помощью современных технологий программирования
ПК-2 - способность и готовность настраивать, тестировать и осуществлять проверку вычислительной техники и программных средств	Знает	способы тестирования программных средств, в том числе, разработанных самостоятельно
	Умеет	находить и исправлять ошибки в тестируемых программных продуктах
	Владеет	навыками настройки вычислительной техники для работы в интегрированной среде разработки, навыками тестирования и проверки программных продуктов
ПК-3 – способность и готовность демонстрировать знания современных языков программирования, операционных систем, офисных приложений, информационно-телекоммуникационной сети "Интернет", способов и механизмов управления данными, принципов организации, состава и схемы работы операционных систем	Знает	принципы работы современных операционных систем, современные языки программирования (в частности, язык Python и его библиотеки NumPy, SciPy, Matplotlib, Pandas, Sklearn)
	Умеет	применять знания современных языков программирования и принципов работы современных операционных систем для разработки программных продуктов
	Владеет	навыками разработки программ и реализации методы обработки научных данных на современных языках программирования

№ п/п	Контролируемые разделы дисциплины	Коды и этапы формирования компетенций	Оценочные средства - наименование		
				текущий контроль	промежуточная аттестация
1	Архитектура компьютерной системы	ОПК-2	Знает	Собеседование (УО-1)	Зачёт, вопросы 1-10
			Умеет	Проект (ПР-9)	Зачет, проект 1-9
			Владеет	Проект (ПР-9)	Зачет, проект 1-9
		ПК-2	Знает	Собеседование (УО-1)	Зачёт, вопросы 1-10
			Умеет	Проект (ПР-9)	Зачет, проект 1-9
			Владеет	Проект (ПР-9)	Зачет, проект 1-9
		ПК-3	Знает	Собеседование (УО-1)	Зачёт, вопросы 1-10
			Умеет	Проект (ПР-9)	Зачет, проект 1-9
			Владеет	Проект (ПР-9)	Зачет, проект 1-9
2	Коммуникационные сети	ОПК-2	Знает	Собеседование (УО-1)	Зачёт, вопросы 11-15
			Умеет	Проект (ПР-9)	Зачет, проект 10-13
			Владеет	Проект (ПР-9)	Зачет, проект 10-13
		ПК-2	Знает	Собеседование (УО-1)	Зачёт, вопросы 11-15
			Умеет	Проект (ПР-9)	Зачет, проект 10-13
			Владеет	Проект (ПР-9)	Зачет, проект 10-13
		ПК-3	Знает	Собеседование (УО-1)	Зачёт, вопросы 11-15
			Умеет	Проект (ПР-9)	Зачет, проект 10-13
			Владеет	Проект (ПР-9)	Зачет, проект 10-13
3	Язык программирования Python и компьютерная графика	ОПК-2	Знает	Собеседование (УО-1)	Экзамен, вопросы 17-23
			Умеет	Проект (ПР-9)	Экзамен, проект 1-8
			Владеет	Проект (ПР-9)	Экзамен, проект 1-8
		ПК-2	Знает	Собеседование (УО-1)	Экзамен, вопросы 17-23
			Умеет	Проект (ПР-9)	Экзамен, проект 1-8

			Владеет	Проект (ПР-9)	Экзамен, проект 1-8
		ПК-3	Знает	Собеседование (УО-1)	Экзамен, вопросы 17-23
			Умеет	Проект (ПР-9)	Экзамен, проект 1-8
			Владеет	Проект (ПР-9)	Экзамен, проект 1-8
4	Машинное обучение	ОПК-2	Знает	Собеседование (УО-1)	Экзамен, вопросы 24-28
			Умеет	Проект (ПР-9)	Экзамен, проект 9-17
			Владеет	Проект (ПР-9)	Экзамен, проект 9-17
		ПК-2	Знает	Собеседование (УО-1)	Экзамен, вопросы 24-28
			Умеет	Проект (ПР-9)	Экзамен, проект 9-17
			Владеет	Проект (ПР-9)	Экзамен, проект 9-17
		ПК-3	Знает	Собеседование (УО-1)	Экзамен, вопросы 24-28
			Умеет	Проект (ПР-9)	Экзамен, проект 9-17
			Владеет	Проект (ПР-9)	Экзамен, проект 9-17

Зачётно-экзаменационные материалы

Вопросы для подготовки к зачету

по дисциплине «Операционные системы, сети, компьютерная графика»

Модуль 1

1. Определение ОС.
2. Функции ОС.
3. Особенности ОС для различных классов компьютерных систем.
4. Управление прерываниями, памятью, вводом-выводом.
5. Кэширование.
6. Основные сведения о процессах.
7. Потоки. Основные операции с потоками.
8. Способы реализации потоков.
9. Взаимодействие процессов.
10. Архитектура ОС UNIX.
11. Распределённые и сетевые системы.
12. Топологии и типы сетей.
13. Сетевые протоколы.
14. Протокол TCP/IP.
15. Подводный Интернет. Прокладка кабелей.

Модуль 2

17. Язык программирования Python. Основные операторы. Синтаксис, семантика.
18. Язык программирования Python. Списки, словари, кортежи, множества.
19. Язык программирования Python. Функции.
20. Язык программирования Python. Работа с компьютерной графикой.
21. Язык программирования Python. Библиотека Numpy.
22. Язык программирования Python. Библиотека Scipy.
23. Язык программирования Python. Библиотека Pandas.

24. Язык программирования Python. Реализация основных методов машинного обучения.
25. Решающие деревья.
26. Метод опорных векторов.
27. Выбор метрики.
28. Логистическая регрессия.

Комплекты оценочных средств для текущей аттестации

Вопросы для собеседования

по дисциплине «Операционные системы, сети, компьютерная графика»

Модуль 1

1. Определение ОС.
2. Функции ОС.
3. Особенности ОС для различных классов компьютерных систем.
4. Управление прерываниями, памятью, вводом-выводом.
5. Кэширование.
6. Основные сведения о процессах.
7. Потоки. Основные операции с потоками.
8. Способы реализации потоков.
9. Взаимодействие процессов.
10. Архитектура ОС UNIX.
11. Распределённые и сетевые системы.
12. Топологии и типы сетей.
13. Сетевые протоколы.
14. Протокол TCP/IP.
15. Подводный Интернет. Прокладка кабелей.

Модуль 2

17. Язык программирования Python. Основные операторы. Синтаксис, семантика.
18. Язык программирования Python. Списки, словари, кортежи, множества.
19. Язык программирования Python. Функции.
20. Язык программирования Python. Работа с компьютерной графикой.
21. Язык программирования Python. Библиотека Numpy.
22. Язык программирования Python. Библиотека Scipy.
23. Язык программирования Python. Библиотека Pandas.

24. Язык программирования Python. Реализация основных методов машинного обучения.
25. Решающие деревья.
26. Метод опорных векторов.
27. Выбор метрики.
28. Логистическая регрессия.

Критерии оценки:

✓ 100-86 баллов - если ответ показывает глубокое и систематическое знание всего программного материала и структуры конкретного вопроса, а также основного содержания и новаций лекционного курса по сравнению с учебной литературой. Студент демонстрирует отчетливое и свободное владение концептуально-понятийным аппаратом, научным языком и терминологией соответствующей научной области. Знание основной литературы и знакомство с дополнительно рекомендованной литературой. Логически корректное и убедительное изложение ответа.

✓ 85-76 - баллов - знание узловых проблем программы и основного содержания лекционного курса; умение пользоваться концептуально-понятийным аппаратом в процессе анализа основных проблем в рамках данной темы; знание важнейших работ из списка рекомендованной литературы. В целом логически корректное, но не всегда точное и аргументированное изложение ответа.

✓ 75-61 - балл – фрагментарные, поверхностные знания важнейших разделов программы и содержания лекционного курса; затруднения с использованием научно-понятийного аппарата и терминологии учебной дисциплины; неполное знакомство с рекомендованной литературой; частичные затруднения с выполнением предусмотренных программой заданий; стремление логически определенно и последовательно изложить ответ.

✓ 60-50 баллов – незнание, либо отрывочное представление о данной проблеме в рамках учебно-программного материала; неумение использовать понятийный аппарат; отсутствие логической связи в ответе.

Итоговая шкала оценивания

Менее 60 баллов	незачтено	неудовлетворительно
От 61 до 75 баллов	зачтено	удовлетворительно
От 76 до 85 баллов	зачтено	хорошо
От 86 до 100 баллов	зачтено	отлично

Темы проектов

по дисциплине «Инструментальные средства обработки и управления данными»

Модуль 1

1. Функции ОС.
2. Особенности ОС для различных классов компьютерных систем.
3. Управление прерываниями, памятью, вводом-выводом.
4. Кэширование.
5. Основные сведения о процессах.
6. Потоки. Основные операции с потоками.
7. Способы реализации потоков.
8. Взаимодействие процессов.
9. Архитектура ОС UNIX.
10. Распределённые и сетевые системы.
11. Топологии и типы сетей.
12. Сетевые протоколы.
13. Протокол TCP/IP.

Модуль 2

1. Язык программирования Python. Расчёт пенсионных накоплений.
2. Язык программирования Python. Проект «Диофантовы уравнения и MacDonalds».
3. Адаптивный тренажёр Python.
4. Линейная алгебра и язык скриптов Python.
5. Работа с массивами в NumPy.
6. Возможности библиотеки SciPy.
7. Графическая обработка и визуализация данных (Matplotlib).
8. Сетевой веб-сервис на Python.
9. Задача распознавания спама в текстовых сообщениях.
10. Предобработка данных в Pandas.

11. Важность признаков.
12. Выбор числа соседей.
13. Выбор метрики.
14. Нормализация признаков.
15. Метод опорных векторов.
16. Анализ текстов.
17. Логистическая регрессия.

Критерии оценки:

✓ 100-86 баллов - если ответ показывает глубокое и систематическое знание всего программного материала и структуры конкретного вопроса, а также основного содержания и новаций лекционного курса по сравнению с учебной литературой. Студент демонстрирует отчетливое и свободное владение концептуально-понятийным аппаратом, научным языком и терминологией соответствующей научной области. Знание основной литературы и знакомство с дополнительно рекомендованной литературой. Логически корректное и убедительное изложение ответа.

✓ 85-76 - баллов - знание узловых проблем программы и основного содержания лекционного курса; умение пользоваться концептуально-понятийным аппаратом в процессе анализа основных проблем в рамках данной темы; знание важнейших работ из списка рекомендованной литературы. В целом логически корректное, но не всегда точное и аргументированное изложение ответа.

✓ 75-61 балл – фрагментарные, поверхностные знания важнейших разделов программы и содержания лекционного курса; затруднения с использованием научно-понятийного аппарата и терминологии учебной дисциплины; неполное знакомство с рекомендованной литературой; частичные затруднения с выполнением предусмотренных программой заданий; стремление логически определено и последовательно изложить ответ.

✓ 60-50 баллов – незнание, либо отрывочное представление о данной проблеме в рамках учебно-программного материала; неумение использовать понятийный аппарат; отсутствие логической связи в ответе.

Итоговая шкала оценивания

Менее 60 баллов	незачтено	неудовлетворительно
От 61 до 75 баллов	зачтено	удовлетворительно
От 76 до 85 баллов	зачтено	хорошо
От 86 до 100 баллов	зачтено	отлично

Описание показателей и критериев оценивания компетенций, шкал оценивания

Критерии оценки собеседования

✓ 100-86 баллов - если ответ показывает глубокое и систематическое знание всего программного материала и структуры конкретного вопроса, а также основного содержания и новаций курса по сравнению с учебной литературой. Студент демонстрирует отчетливое и свободное владение концептуально-понятийным аппаратом, научным языком и терминологией соответствующей научной области. Знание основной литературы и знакомство с дополнительно рекомендованной литературой. Логически корректное и убедительное изложение ответа.

✓ 85-76 - баллов - знание узловых проблем программы и основного содержания курса; умение пользоваться концептуально-понятийным аппаратом в процессе анализа основных проблем в рамках данной темы; знание важнейших работ из списка рекомендованной литературы. В целом логически корректное, но не всегда точное и аргументированное изложение ответа.

✓ 75-61 - балл – фрагментарные, поверхностные знания важнейших разделов программы и содержания курса; затруднения с использованием научно-понятийного аппарата и терминологии учебной дисциплины; неполное знакомство с рекомендованной литературой; частичные затруднения с выполнением предусмотренных программой заданий; стремление логически определено и последовательно изложить ответ.

✓ 60-50 баллов – незнание, либо отрывочное представление о данной проблеме в рамках учебно-программного материала; неумение использовать понятийный аппарат; отсутствие логической связи в ответе.

Итоговая шкала оценивания

Менее 60 баллов	незачтено	неудовлетворительно
От 61 до 75 баллов	зачтено	удовлетворительно
От 76 до 85 баллов	зачтено	хорошо
От 86 до 100 баллов	зачтено	отлично

Методические рекомендации, определяющие процедуры оценивания результатов освоения дисциплины

Текущая аттестация студентов. Текущая аттестация студентов по дисциплине «Операционные системы, сети, компьютерная графика» проводится в соответствии с локальными нормативными актами ДВФУ и является обязательной.

Текущая аттестация по дисциплине «Операционные системы, сети, компьютерная графика» проводится в форме собеседования и выполнения проектных работ, практических заданий, лабораторных работ и осуществляется ведущим преподавателем.

Объектами оценивания выступают:

- степень усвоения теоретических знаний - оценивается в форме собеседования;
- уровень овладения практическими умениями и навыками – оценивается в форме выполнения лабораторных работ, практических заданий, работы над проектами.

Промежуточная аттестация студентов. Промежуточная аттестация студентов по дисциплине «Операционные системы, сети, компьютерная графика» проводится в соответствии с локальными нормативными актами ДВФУ и является обязательной.

По дисциплине предусмотрены зачёт в 5 семестре и экзамен в 6 семестре, которые проводятся в устной форме.

Критерии выставления оценки студенту на зачёте/ экзамене по дисциплине «Операционные системы, сети, компьютерная графика»

Баллы (рейтинговой оценки)	Оценка зачета/ экзамена (стандартная)	Требования к сформированным компетенциям
----------------------------------	---	---

	<p><i>«зачтено»/ «отлично»</i></p>	<p>Оценка «отлично» выставляется студенту, если он глубоко и прочно усвоил программный материал, исчерпывающе, последовательно, четко и логически стройно его излагает, умеет тесно увязывать теорию с практикой, свободно справляется с задачами, вопросами и другими видами применения знаний, причем не затрудняется с ответом при видоизменении заданий, правильно обосновывает принятое решение, владеет разносторонними навыками и приёмами выполнения лабораторных работ; знает современные технологии программирования на языке Python; умеет разрабатывать программные продукты с помощью современных технологий программирования; владеет эффективными методами разработки программных продуктов с помощью современных технологий программирования; знает технологии отладки и тестирования прикладного программного обеспечения; умеет отлаживать и тестировать прикладное программное обеспечение; владеет методами отладки и тестирования прикладного программного обеспечения; знает способы тестирования программных средств, в том числе, разработанных самостоятельно; умеет находить и исправлять ошибки в тестируемых программных продуктах; владеет навыками настройки вычислительной техники для работы в интегрированной среде разработки, навыками тестирования и проверки программных продуктов; знает современные языки программирования (в частности, язык Python); умеет применять знания современных языков программирования для разработки программных продуктов; владеет навыками разработки программ на современных языках программирования.</p>
	<p><i>«зачтено»/ «хорошо»</i></p>	<p>Оценка «хорошо» выставляется студенту, если он твердо знает материал, грамотно и по существу излагает его, не допуская существенных неточностей в ответе на вопрос, правильно применяет теоретические положения при решении практических вопросов и задач, владеет необходимыми навыками и приемами их выполнения; знает современные технологии программирования на языке Python; умеет разрабатывать программные продукты с помощью</p>

		<p>современных технологий программирования; владеет эффективными методами разработки программных продуктов с помощью современных технологий программирования; умеет находить и исправлять ошибки в тестируемых программных продуктах; владеет навыками настройки вычислительной техники для работы в интегрированной среде разработки; знает современные языки программирования (в частности, язык программирования Python); умеет применять знания современных языков программирования для разработки программных продуктов; владеет навыками разработки программ на современных языках программирования.</p>
	<p><i>зачтено»/ «удовлетворительно»</i></p>	<p>Оценка «удовлетворительно» выставляется студенту, если он имеет знания только основного материала, но не усвоил его деталей, допускает неточности, недостаточно правильные формулировки, нарушения логической последовательности в изложении программного материала, испытывает затруднения при выполнении лабораторных работ; частично владеет навыками разработки программ на современных языках программирования.</p>
	<p><i>«не зачтено»/ «неудовлетворительно»</i></p>	<p>Оценка «неудовлетворительно» выставляется студенту, который не знает значительной части программного материала, допускает существенные ошибки, неуверенно, с большими затруднениями выполняет лабораторные работы, практически не владеет навыками разработки программ на современных языках программирования. Как правило, оценка «неудовлетворительно» ставится студентам, которые не могут продолжить обучение без дополнительных занятий по соответствующей дисциплине.</p>