МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РЕСПУБЛИКИ БЕЛАРУСЬ

БЕЛОРУССКИЙ НАЦИОНАЛЬНЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

## Факультет информационных технологий и робототехники

Кафедра программного обеспечения информационных систем

и технологий

**Отчет**

**по лабораторной работе № 4**

по дисциплине: ”Системное программирование”

на тему: ***”*Cкрипт-проект-диалоги*”***

Выполнил**:** студенты группы *10702121* Агеенко А. К.

Бродко Г. Г.

Высоцкий М. Л.

Градусов А. Р.

Принял**:** пр. Давыденко Н. В.

Минск 2023

# Лабораторная работа №4.

**Цель работы:** Закрепить на практике принципы создания проектов с помощью скриптов Shell, освоить средства примитивного графического интерфейса в сприктах Shell.

## Задание 1

1. Cделать пример с двумя вложенными диалогами типа YesNo.

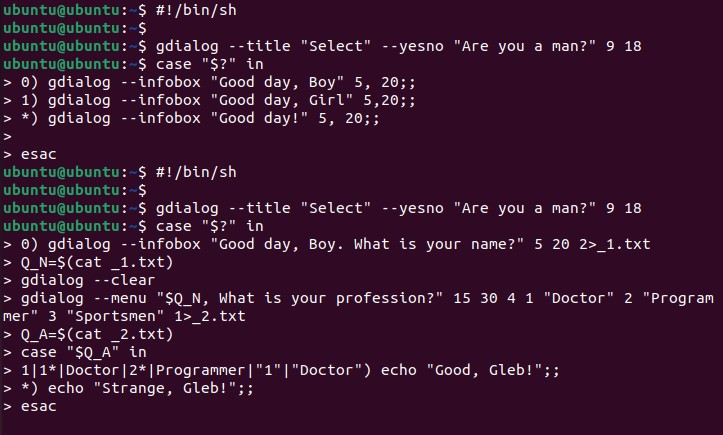
2. Cделать пример, запрашивающий сначала имя человека, а потом профессию. Вывести имя+профессию, прочитанную в диалоге.

3. Вывести список с названиями валют. После выбора валюты система должна вывести ее котировку.

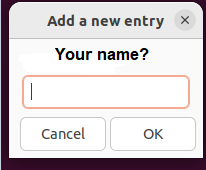
4. Сделать оконное меню валют к 3 пункту.

5. Добавить выход из программы.

**Решение**

1.Пишем скрипт с двумя вложенными диалогами типа YesNo сразу вместе с элементами для пункта 2.

2.Запрашиваем на ввод имя человека



3.Запрашиваем у пользователя информацию о его професии.



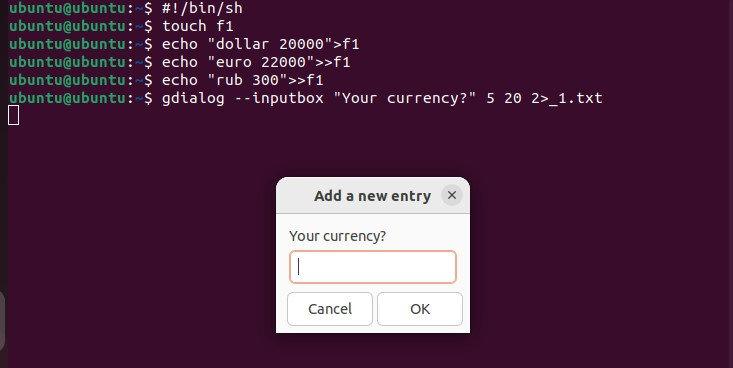
4.Реализовываем систему ответа на выбранный вариант



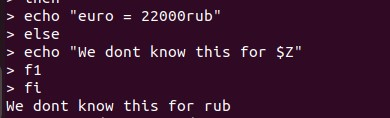
**2.**

1. Прописываем скрипт и спрашиваем у пользователя валюту, с которой он хочет работать.

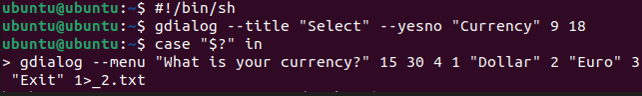


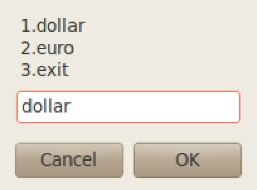


1. При вводе euro получаем котировку, при вводе незаданной валюты, получаем реализованное сообщение о том, что введене не так валюты

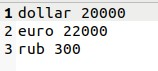


**3.**

1. Делаем оконное меню для валют и сразу реализовываем опцию выхода 



1. Создаем отдельное окно для выхода и добавочное окно катировки



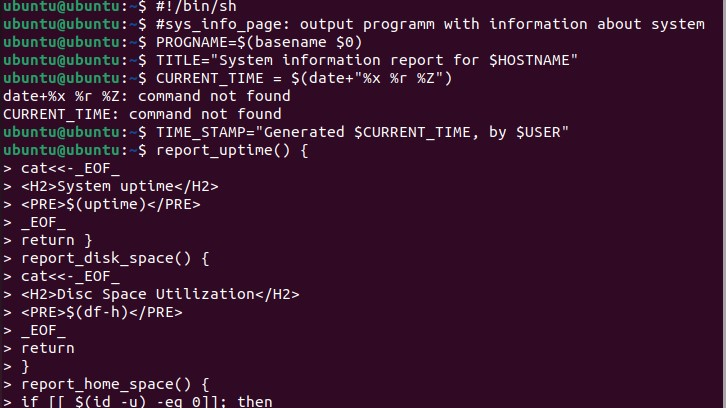


**Задание 2.**

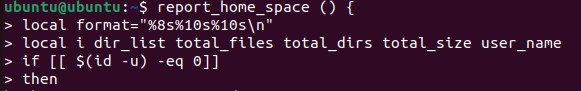
6. Создайте копию проекта из упражнения 3.

7. Добавьте туда файловый вывод

**4.** Создаем копию проекта



**5.** Добавляем туда файловый вывод



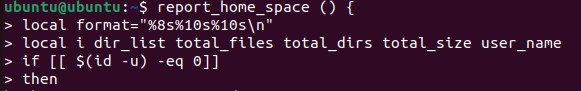
**Задание 3.**

8. Довести проект до совершенства, добавив выбор функций

9. Сделать так, чтобы информация хранилась в html файле и открывалась в браузере

**8.**

1. Добавляем домашнюю страницу для выбора функций



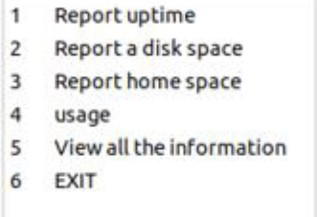
1. Добавляем выбор базовых функций



1. Реализовываем интерфейс для вывод в html документ



1. Получаем итоговое меню для выбора функций



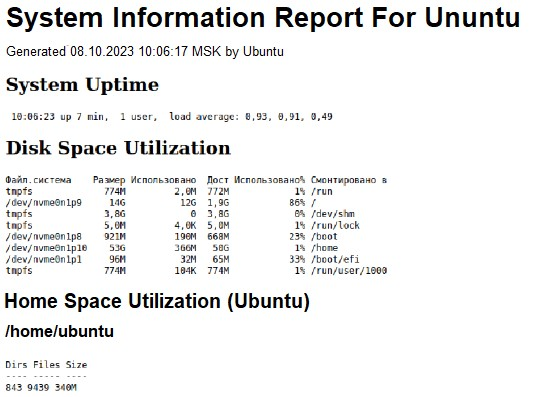
1. Подтверждаем запись в html файл



1. Открываем файл в браузере по нашему пути



1. Наблюдаем за итоговым результатом в браузере



Контрольные вопросы:

1. **Что такое переменные и константы, чем они отличаются? Как их различает командная оболочка?**

Переменные и константы - это символические имена, которые используются для хранения значений в программировании.

Основное отличие между переменными и константами заключается в том, что значение переменной может изменяться в течение выполнения программы, тогда как значение константы остается постоянным и неизменным.

Командная оболочка различает переменные и константы по способу их определения и использования. В большинстве командных оболочек переменные обычно определяются с помощью оператора присваивания, например:

variable=5

Константы, с другой стороны, часто определяются при помощи ключевых слов или специальных функций, которые указывают, что значение не должно изменяться. В некоторых командных оболочках можно создать константу, используя оператор readonly, например:

readonly constant=10

Командная оболочка также может предоставлять специальные синтаксические конструкции для работы с переменными и константами, такие как подстановка значений переменных, операции присваивания и другие функции для работы с данными.

1. **Как командная оболочка различает тип значения переменных?**

Командная оболочка не различает явным образом тип значения переменных, так как она по умолчанию рассматривает все значения как строки, то есть символьные последовательности. Это означает, что независимо от того, введено ли значение в виде числа, текста или другого типа данных, командная оболочка будет рассматривать его как строку.

Однако в командных оболочках есть возможность выполнять преобразование типов данных для значений переменных, чтобы использовать их в математических операциях или других контекстах.

Например, в bash можно использовать операторы для выполнения математических операций над числами, но перед этим необходимо явно указать, что переменная содержит числовое значение, например:

num1=5

num2=10

sum=$((num1 + num2))

echo $sum # Выводит 15

Здесь $((...)) - это специальный синтаксис для выполнения математических операций в bash. Он позволяет командной оболочке распознать, что значения num1 и num2 являются числами, а не строками.

Таким образом, хотя командная оболочка рассматривает значения переменных по умолчанию как строки, различные синтаксические конструкции позволяют явно указывать типы данных и выполнять преобразования типов для удобной работы с разными типами значений.

1. **Какое типичное соглашение о написании имен переменных и констант принято программистами? Для чего это делается?**

Типичное соглашение о написании имен переменных и констант, которое принято среди программистов, называется стилем именования переменных и следует определенным правилам. Некоторые распространенные соглашения включают:

1. Имена переменных обычно записываются в нижнем регистре (например, `count`, `name`) и могут содержать буквы, цифры и символ подчеркивания `\_`. Пробелы не разрешены.

2. Для разделения слов в именах переменных обычно используется принятый способ называемый "snake\_case", где слова разделяются символом подчеркивания (например, `first\_name`, `total\_count`).

3. Имена констант традиционно записываются в верхнем регистре (например, `MAX\_COUNT`, `PI`) и также могут содержать буквы, цифры и символ подчеркивания.

4. Если переменная является локальной и используется только внутри функции или блока кода, то может использоваться некоторое соглашение об использовании "lowerCamelCase", где первое слово начинается с маленькой буквы, а каждое следующее слово начинается с заглавной буквы (например, `totalCount`, `firstName`).

Соглашения о именах переменных и констант используются для улучшения читаемости кода и облегчения его понимания. Они помогают сделать код более ясным, согласованным и удобным для сопровождения. Стиль именования переменных отлично подходит, чтобы отличить переменные от констант, понять, что эти символы имеют смысл и соблюдать единообразие во всем коде программы.

1. **Что такое переменные окружения? Назовите команды с помощью которых можно получить значения переменных окружения.**

Переменные окружения - это переменные, которые устанавливаются в операционной системе и доступны для всех процессов, работающих на этой системе. Они содержат информацию о системных настройках, текущем состоянии системы и других параметрах, используемых программами и командами.

Для получения значений переменных окружения доступны различные команды, в зависимости от операционной системы.

Вот некоторые примеры команд для получения значений переменных окружения:

1. В Windows командная оболочка CMD:

- set - выводит список всех переменных окружения и их значений.

- echo %VariableName% - выводит значение конкретной переменной окружения, заменив VariableName на имя переменной.

2. В Windows PowerShell:

- Get-ChildItem Env: - выводит список всех переменных окружения и их значений.

- Get-Item Env:VariableName - выводит значение конкретной переменной окружения, заменив VariableName на имя переменной.

3. В Unix/Linux командной оболочке (например, bash):

- env - выводит список всех переменных окружения и их значений.

- echo $VariableName - выводит значение конкретной переменной окружения, заменив VariableName на имя переменной.

4. В macOS командной оболочке (также используется bash):

- env или printenv - выводит список всех переменных окружения и их значений.

- echo $VariableName - выводит значение конкретной переменной окружения, заменив VariableName на имя переменной.

При использовании этих команд вы можете получить доступ к информации, хранящейся в переменных окружения, такой как путь к исполняемым файлам, настройки локали, домашний каталог и многие другие.

1. **Что такое пользовательские переменные и для чего они используются?**

Пользовательские переменные - это переменные, созданные и использованные в рамках конкретной программы или сценария. Они не являются частью операционной системы или переменных окружения, а создаются и управляются самой программой или сценарием.

Пользовательские переменные используются для хранения и обработки определенных значений, которые могут быть использованы в различных частях программы или сценария. Они могут быть использованы для передачи данных между функциями и блоками кода, для сохранения временных результатов вычислений и для хранения конфигурационных параметров.

Преимущества использования пользовательских переменных:

1. Удобство: Пользовательские переменные позволяют работать с данными подходящим способом и использовать их в различных частях программы.

2. Читаемость и понятность: Правильное именование пользовательских переменных помогает понять назначение переменной в программе без необходимости изучения всего кода.

3. Гибкость: Пользовательские переменные могут быть легко изменены или переназначены при необходимости менять значения в коде.

4. Безопасность: Использование пользовательских переменных может помочь избежать жесткого закодирования значений, что облегчает изменение программы или сценария в будущем.

Хотя пользовательские переменные специфичны для каждой программы или сценария, они предоставляют много гибкости и контроля, позволяя программистам более удобно управлять данными и привносить логику и структуру в программный код.

1. **Что такое область видимости переменных? Какие типы области видимости переменных существуют?**

Область видимости переменных - это часть программы, в которой переменная может быть использована или доступна. Область видимости определяет контекст, в котором переменная существует и может быть обращена.

Существует несколько типов области видимости переменных:

1. Глобальная область видимости: Переменная, объявленная в глобальной области видимости, доступна из любой части программы, включая все функции и блоки кода. Глобальные переменные могут быть использованы и изменены в любом месте программы. Они обычно объявляются в начале программы и сохраняют свои значения на протяжении всего времени выполнения программы.

2. Локальная область видимости: Переменная, объявленная внутри функции или блока кода, имеет локальную область видимости. Локальные переменные видимы только в пределах функции или блока, в котором они объявлены. Они могут быть использованы и изменены только внутри той же функции или блока. При выполнении блоков кода для циклов, условий или других структур, локальная область видимости ограничивается соответствующим блоком.

3. Параметры функций: Параметры функций также имеют область видимости, которая ограничена телом функции. Они являются локальными переменными, которые принимают значения аргументов, переданных функции при ее вызове.

Область видимости переменных влияет на доступность и значение переменных в разных частях программы, а также на то, как они взаимодействуют друг с другом. Правильное управление областью видимости помогает избежать конфликтов и позволяет программистам создавать более структурированный и надежный код.

1. **Что такое конфликт имен переменных и как решается эта проблема?**

Конфликт имен переменных возникает, когда в одной области видимости программы используется несколько переменных с одинаковыми именами. Это может привести к непредсказуемому поведению программы или ошибкам в результате неправильного использования значений переменных.

Для решения проблемы конфликта имен переменных можно использовать следующие подходы:

1. Изменение имен переменных: Простейший способ решить конфликт имен - это переименовать переменные, чтобы они были уникальными в каждой области видимости. Это помогает явно указать, какая переменная используется в определенном контексте.

2. Использование разных областей видимости: Если переменные с одинаковыми именами требуются для разных частей программы, то создание переменных с разными областями видимости может быть полезным. Таким образом, каждая переменная будет существовать только в нужной области и не будет конфликта имен.

3. Использование префиксов или пространств имен: Для устранения конфликта имен можно добавить префиксы или используйте пространства имен для переменных. Например, если у вас есть переменная count в разных частях программы, вы можете использовать user\_count и system\_count для различения.

4. Использование объектно-ориентированного подхода: Если вы работаете в объектно-ориентированном языке программирования, вы можете использовать классы и экземпляры классов для изолирования переменных с одинаковыми именами в разных объектах.

Важно придерживаться правильного стиля именования переменных и быть внимательным, чтобы избежать конфликта имен. Решение проблемы конфликта имен переменных помогает сделать программный код более читаемым, легким в понимании и предсказуемым.

1. **Что такое функции командной оболочки?**

Функции командной оболочки (shell functions) - это фрагменты кода, которые объединяют последовательность команд для выполнения определенной задачи. Они являются основным механизмом структурирования и повторного использования кода в командных оболочках.

Функции командной оболочки обычно объявляются с использованием ключевого слова function, за которым следует имя функции и открывающаяся фигурная скобка {. Затем идет последовательность команд, которые должны быть выполнены при вызове данной функции. Функция может принимать аргументы, которые могут быть переданы при ее вызове.

Преимущества использования функций командной оболочки:

1. Повторное использование кода: Функции позволяют группировать код, что облегчает его повторное использование в разных частях скрипта или программы.

2. Структурирование кода: Функции помогают логически структурировать код, делая его более понятным и

1. **Что такое код завершения программы?**

Код завершения программы, также называемый кодом возврата или выходным кодом, представляет собой целое число, возвращаемое программой операционной системе после ее выполнения. Он указывает операционной системе результат выполнения программы, например, успешное завершение, ошибку или иное состояние. Код завершения может быть проверен другими программами или скриптами для принятия соответствующих действий. Коды завершения обычно согласованы соглашениями в операционной системе, но могут отличаться в зависимости от платформы или языка программирования.

1. **Какой командой и каким средство в скриптах обеспечивается интерактивность программы?**

В скриптах интерактивность программы может быть обеспечена с помощью команды input() и функции print().

Команда input() используется для чтения пользовательского ввода со стандартного ввода. Она ожидает ввода от пользователя и возвращает введенные данные в виде строки. Пример использования:

name = input("Введите ваше имя: ")

print("Привет, " + name)

Функция print() используется для вывода данных на стандартный вывод. Она принимает один или несколько аргументов и выводит их на экран. Пример использования:

age = 25

print("Ваш возраст:", age)

При сочетании команды input() и функции print() в скриптах можно создавать интерактивные программы, в которых пользователь может вводить данные и видеть результаты на экране.

1. **С помощью каких утилит реализуются средства примитивного графического интерфейса в Linux-скриптах?**

В Linux-скриптах для реализации средств примитивного графического интерфейса часто используются следующие утилиты:

1. Zenity: Zenity позволяет создавать диалоговые окна с кнопками, ползунками, текстовыми полями и т. д. Пример использования Zenity:

zenity --info --text="Привет, мир!"

2. Yad: Yad (Yet Another Dialog) является форком Zenity и предоставляет дополнительные возможности для создания диалоговых окон и виджетов. Пример использования Yad:

yad --title="Привет, мир!" --text="Привет, мир!" --button="Закрыть":0

3. Dialog: Dialog предоставляет более широкий спектр виджетов и позволяет создавать интерактивные меню, ввод данных и т. д. Пример использования Dialog:

dialog --msgbox "Привет, мир!" 10 30

Эти утилиты позволяют скриптам в Linux взаимодействовать с пользователем через графический интерфейс и обрабатывать различные действия и выборы, сделанные пользователем, что делает скрипты более удобными и понятными для использования.

1. **Укажите какими интернет-источниками вы пользовались при изучении и разработке графического окна.**

При изучении и разработке графического окна я использовал следующие интернет-источники:

1. Официальная документация и руководства по использованию инструментов и библиотек разработки графического интерфейса, таких как PyQt, Tkinter, wxPython, GTK+ и других. Данные руководства предоставляют информацию о функциях, классах, методах и примерах использования.

2. Форумы и сообщества разработчиков, такие как Stack Overflow, Reddit, форумы разработчиков языков программирования. Часто можно найти ответы на разные вопросы, статьи, посоветоваться с опытными программистами и получить рекомендации по созданию графических окон.

3. Туториалы и руководства на блогах и веб-сайтах, посвященных разработке программного обеспечения. Эти материалы часто содержат шаг за шагом инструкции, примеры кода и подробное объяснение различных концепций и методов, связанных с созданием графического интерфейса.

4. Официальные документации по спецификациям стандартов, таких как HTML, CSS, JavaScript, для разработки веб-интерфейсов.

Указанные источники помогли мне получить все необходимые знания и навыки для изучения и разработки графических окон.

1. **Какие существуют типы окон примитивного графического интерфейса Linux-cкриптов?**

В Linux-скриптах существует несколько типов окон примитивного графического интерфейса, которые могут быть реализованы с помощью утилит, таких как Zenity, Yad и Dialog. Некоторые из распространенных типов окон включают в себя:

1. Диалоговое окно с информацией (Info Dialog): Отображает информационное сообщение или уведомление без кнопок или возможности взаимодействия с пользователем.

Пример: zenity --info --text="Привет, мир!"

2. Диалоговое окно подтверждения (Confirmation Dialog): Запрашивает у пользователя подтверждение или выбор между двумя вариантами (например, "Да" или "Нет").

Пример: zenity --question --text="Вы согласны?"

3. Диалоговое окно с вводом данных (Entry Dialog): Запрашивает у пользователя ввод данных (текста) с возможностью сохранения этого введенного значения.

Пример: zenity --entry --text="Введите ваше имя:"

4. Диалоговое окно выбора файла или директории (File/Directory Selection Dialog): Позволяет пользователю выбрать файл или директорию на компьютере.

Пример: zenity --file-selection --title="Выбор файла"

5. Диалоговое окно списка или меню выбора (List/Menu Selection Dialog): Предоставляет пользователю список вариантов для выбора.

Пример: zenity --list --title="Выберите цвет" --column="Имя" --column="Цвет" Ян "Красный" Лена "Синий"

Это лишь некоторые из типов окон, доступных для использования в скриптах с помощью утилит примитивного графического интерфейса в Linux.