



Компьютерные технологии в научных исследованиях

```
1 #!/bin/bash
2 #INPUT_SAMPLE_LIST=$1
3 cd /Volumes/PhilDrive_EMS/TestDec7/snv_postp
4
11 . paths.txt
12
30
31 echo "Debug level set for $DEBUG_LEVEL"
32 echo "log found in scripts directory"
33
50 cp $HIGH_SNP_OUT ./
51 cp $LOW_SNP_OUT ./
52 cp $GERM_SNP_OUT ./
53 # echo "${SCRIPT_DIR}/run_somatic_mu
54 if [ $DEBUG_LEVEL
55 then
56 echo "INFO: ${SCR
57 `basename ${LOW_S
58 ${D_BAM_FILE} ${G
59
60 fi
61 ${SCRIPT_DIR}run_somatic_mu
62
```

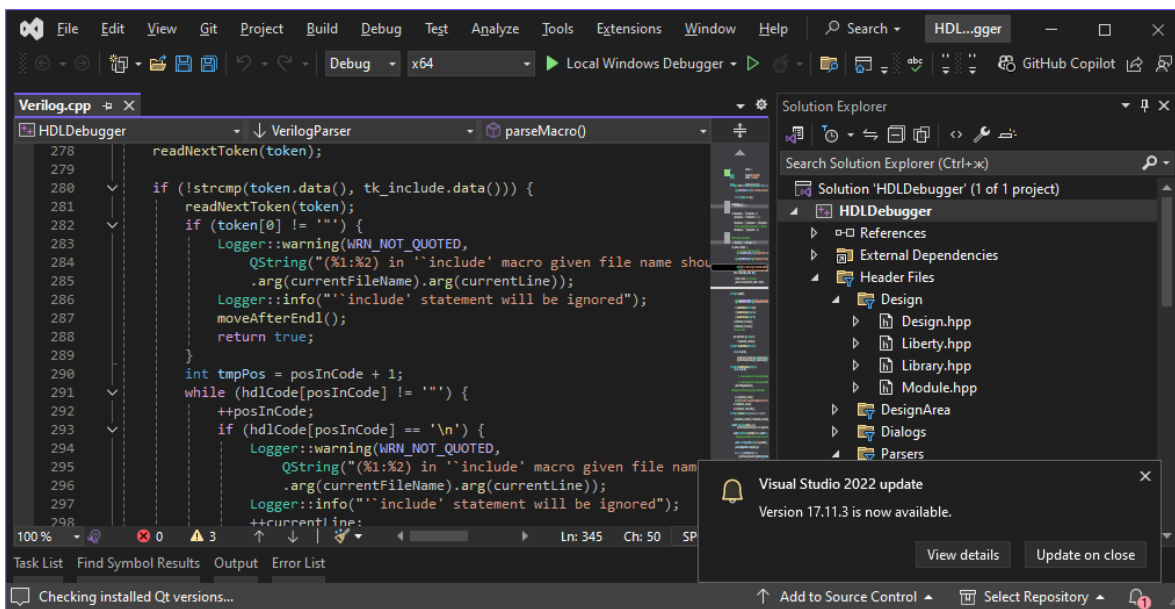


Семинар №1

Скриптовые языки в САПР.

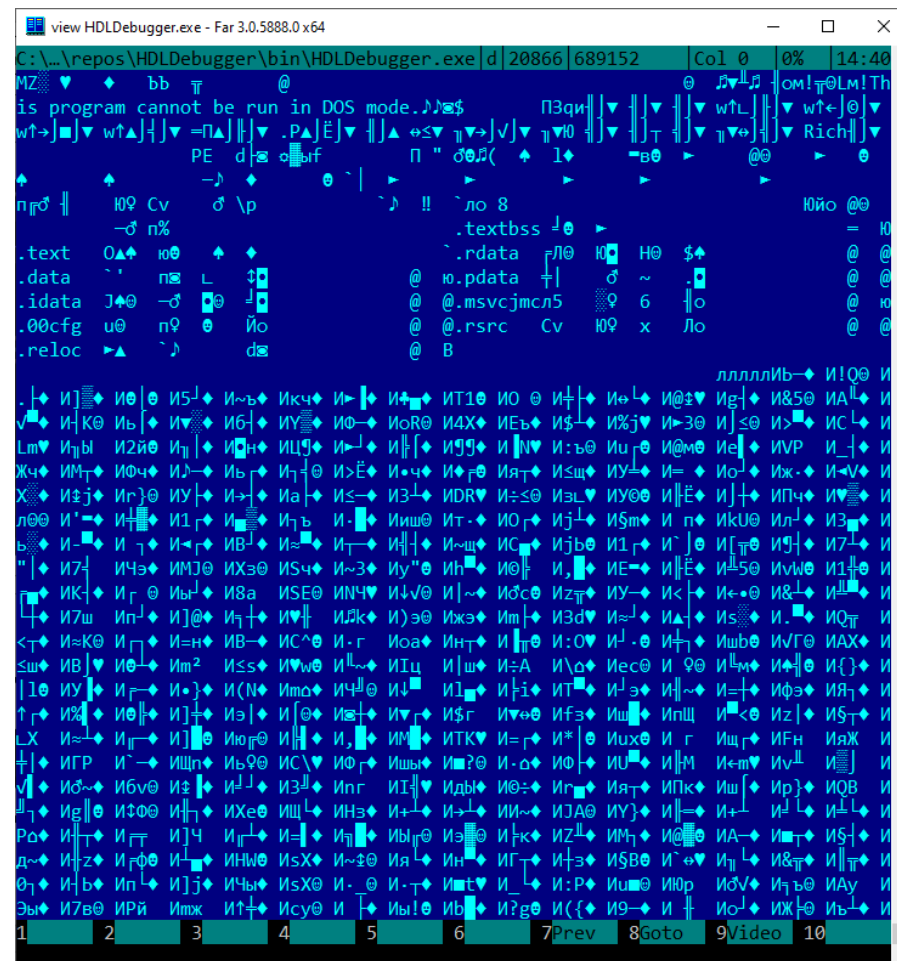
Язык Tcl

Скриптовые языки vs компилируемые языки (1)



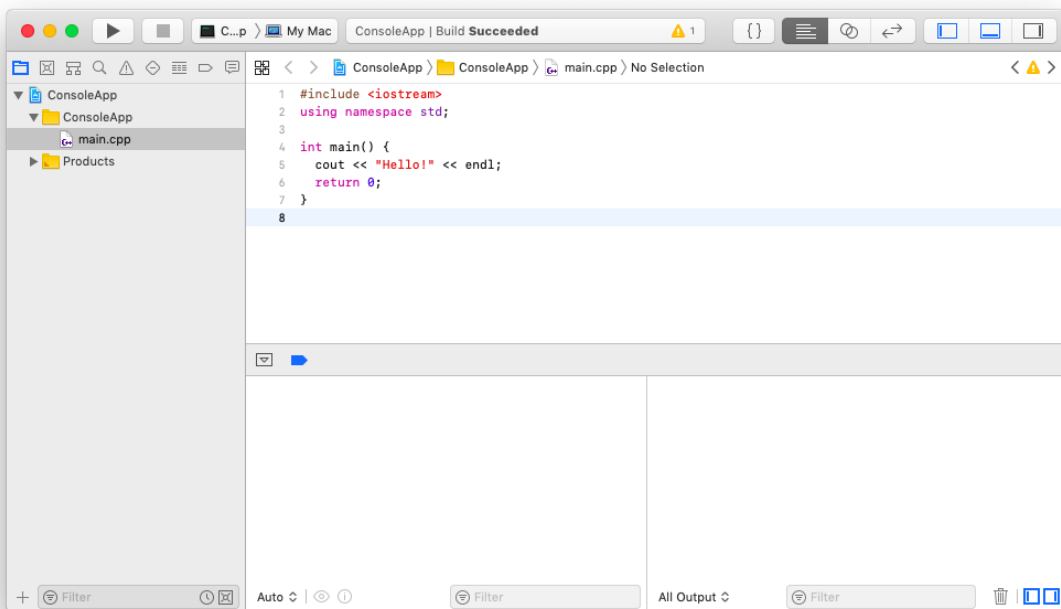
Компиляция и сборка

Бинарный
исполняемый
файл



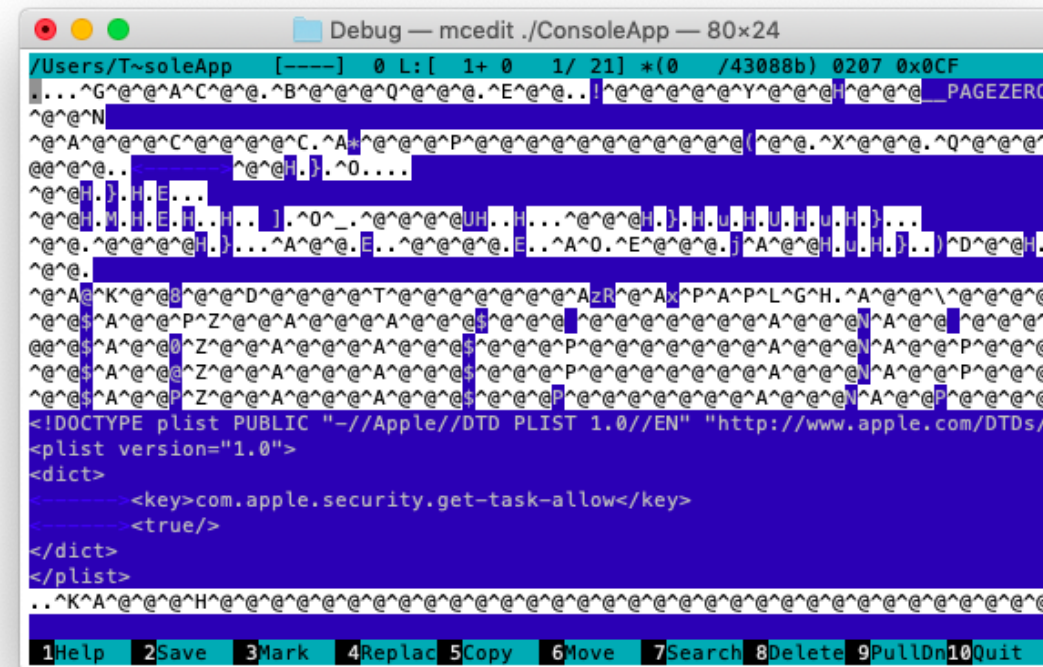
Команды управления
состоянием микропроцессора

Скриптовые языки vs компилируемые языки (2)



Компиляция и сборка

Бинарный
исполняемый
файл



Команды управления
состоянием микропроцессора

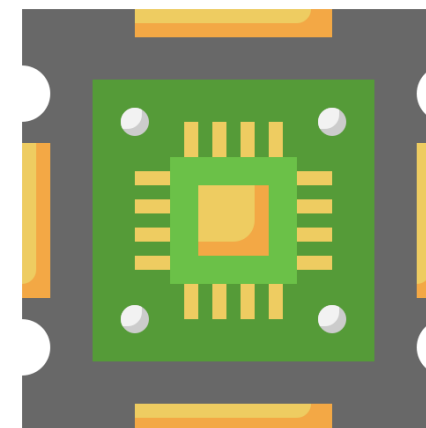
Скриптовые языки vs компилируемые языки (3)

```
F:\python\script_pyqt5.py - Notepad++
File Edit Search View Encoding Language Settings Macro Run Plugins Window ? X
script_pyqt5.py x
1  #!/usr/bin/python3
2  # -*- coding: utf-8 -*-
3
4  import sys
5  from PyQt5.QtWidgets import QApplication, QWidget
6
7  if __name__ == '__main__':
8
9      app = QApplication(sys.argv)
10
11     w = QWidget()
12     w.resize(250, 150)
13     w.move(300, 300)
14     w.setWindowTitle('Simple')
15     w.show()
16
17     sys.exit(app.exec_())
length: Ln:1 Col:1 Sel:0|0 Windows (CR LF) UTF-8 INS
```



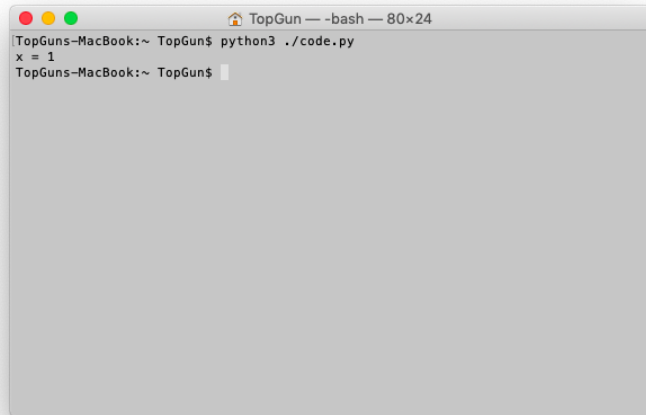
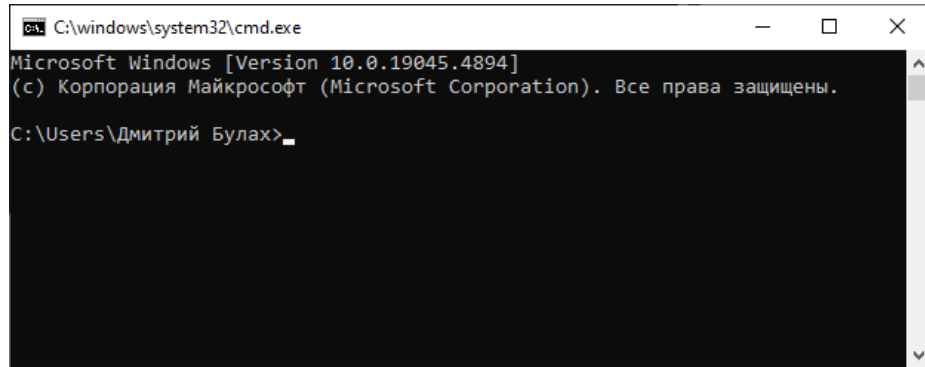
Интерпретатор языка Python

```
C:\WINDOWS\system32\cmd.exe
F:\python>python3.exe ./script_pyqt5.py _
```



Классификация скриптовых языков

Языки командных интерпретаторов



Языки программирования



Языки разметки (условно)



Характеристика скриптовых языков

1. Простота языковых конструкций.
2. Высокая скорость написания простых приложений.
3. Поддержка команд операционной системы на уровне языка.
4. Динамическая типизация данных.
5. Переносимость кода между платформами и архитектурами.
1. Низкая скорость работы кода, связанная с процессом интерпретации кода.
2. Сложность отладки, большая вероятность использования неработоспособного кода.
3. Сложность написания больших проектов, чаще всего скрипты используются как связки или переходники между программами.
4. Отсутствие интерпретаторов для некоторых платформ.
5. Невозможность просто выполнять некоторые элементарные операции, реализуемые в языках программирования высокого уровня.
6. Открытость исходного кода.

Скриптовые языки в САПР

Обычные языки программирования



Свои языки программирования

```
rectCount = lineCount = polygonCount = 0
shapeTypeList = '( "rect" "polygon" "rect" "line" )
foreach( shapeType shapeTypeList
  case( shapeType
    ( "rect"    ++rectCount )
    ( "line"   ++lineCount )
    ( "polygon" ++polygonCount )
    ( t        ++miscCount )
  ) ;case
) ; foreach
=> ( "rect" "polygon" "rect" "line" )
```



Cadence: OceanScript

```
ocnWaveformTool( 'wavescan )  
simulator( 'spectreS )  
design( "/home/topgun/cadence/test_dsn" )  
resultsDir("/home/topgun/cadence/test_dsn/results" )  
temp( 27 )  
analysis('tran ?stop "40n" )  
run()  
selectResult( 'tran )  
plot(getData("/in") getData("/out") )
```



OpenROAD GUI: Tcl/Tk

The screenshot displays the OpenROAD GUI interface. The main window shows a PCB layout with various layers and nets. The 'Display Control' panel on the left lists layers (li1, mcon, met1-5, via1-5) and nets, with checkboxes for visibility. The central canvas shows a grid of layers with a timing report overlaid. The 'Timing Report' window on the right shows a table with columns for 'Capture Cloc', 'Required', 'Arrival', and 'Slack'. The 'Scripting' window at the bottom shows the following output:

```
[INFO]: Setting output delay to: 2.0
[INFO]: Setting input delay to: 2.0
[WARNING STA-0337] port '_VIRTUAL_CLK_' not found.
[INFO]: Setting load to: 0.033442
[INFO]: Setting clock uncertainty to: 0.25
[INFO]: Setting clock transition to: 0.15
[WARNING STA-0559] transition time can not be specified for virtual clocks.
[INFO]: Setting timing derate to: 5.0 %
Reading spefs ...
```

The 'Scripting' window also shows a list of commands: report_d, report_dcalc, report_design_area, report_design_area_metrics, report_disabled_edges. The status bar at the bottom right shows coordinates: 7.169, -2.389.



Открытый маршрут OpenLane: Tcl

```
set ::env(OPENLANE_ROOT) [file dirname [file normalize [info script]]]
set ::env(SCRIPTS_DIR) "$::env(OPENLANE_ROOT)/scripts"

if { [file exists $::env(OPENLANE_ROOT)/install/env.tcl ] } {
    source $::env(OPENLANE_ROOT)/install/env.tcl
}

if { ! [info exists ::env(OPENROAD_BIN) ] } {
    set ::env(OPENROAD_BIN) openroad
}
lappend ::auto_path "$::env(OPENLANE_ROOT)/scripts/"
package require openlane;

proc run_placement_step {args} {
    if { ! [ info exists ::env(PLACEMENT_CURRENT_DEF) ] } {
        set ::env(PLACEMENT_CURRENT_DEF) $::env(CURRENT_DEF)
    } else {
        set ::env(CURRENT_DEF) $::env(PLACEMENT_CURRENT_DEF)
    }

    run_placement
}
```

Язык Tcl (1)



TCL (англ. «Tool Command Language») – долгое время остаётся одним из самых мощных интерпретируемых языков программирования.

Автор: Джон Астераут (John Ousterhout),
профессор информатики Калифорнийского
Университета в Беркли.

Разрабатывается с 1988 года



Черты языка Tcl

1. Программа на Tcl состоит из команд, разделённых символами перевода строки или точками с запятой.
2. Ключевых слов как таковых нет — понятие команды в Tcl аналогично понятию процедуры или функции распространённых языков программирования.
3. В перечень команд языка входят аналоги основных операторов языков программирования.
4. Tcl содержит развитые средства работы с регулярными выражениями, ассоциативными массивами.
5. Является процедурным.

```
#!/usr/bin/tclsh
```

```
# Say hello
```

```
set h "Hello"  
set c ,  
set w "world"  
set exm !
```

```
# Print message on screen
```

```
puts stdout "$h$c $w$exm"
```

Фишки языка Tcl (1)

1. В Tcl данными всех типов, включая код программы, можно манипулировать как строками.

```
set y0 "zzz"  
puts $y0
```

На экран будет выведено:
zzz

```
set x 5  
set y "x"
```

```
puts [expr $$y]
```

На экран будет выведено:
5

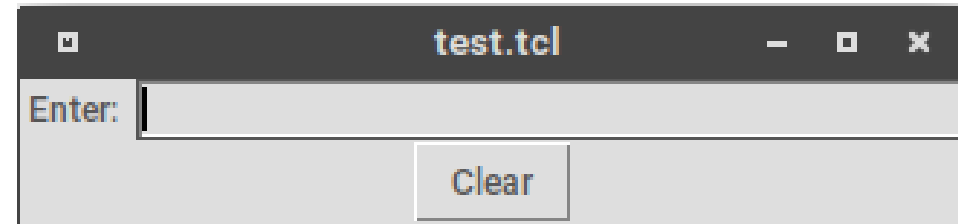
```
set y0 "zzz"  
set one "y"  
set two "0"
```

```
puts [eval "set $one$two"]
```

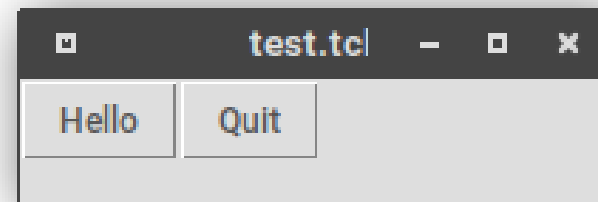
На экран будет выведено:
zzz

Фишки языка Tcl (2)

2. Язык TCL единственный интерпретируемый язык, в состав поставки которого входит собственная библиотека для проектирования графического интерфейса пользователя, Tk.



```
student@localhost:~/tcl> ./test.tcl  
Hello  
█
```



Фишки языка Tcl (3)

3. Для Tcl впервые были разработаны библиотеки для взаимодействия с программами на C++.

```
#include "cpptcl.h"
#include <iostream>

using namespace std;

void hello()
{
    cout << "Hello C++/Tcl!" << endl;
}

CPPTCL_MODULE(mymodule, i)
{
    i.def("hello", hello);
}
```

Можно из C++ выполнять Tcl-код, можно на C++ писать расширения для Tcl.

```
#!/bin/tclsh

load ./mymodule.so

hello
```

Фишки языка Tcl (4)

4. Для Tcl впервые была реализована возможность вызывать код на Tcl из C++.

```
#include "cpptcl.h"
#include <iostream>

using namespace std;

int main()
{
    Tcl_Interp *interp = Tcl_CreateInterp();

    int code;
    code = Tcl_Eval(interp, "set a 1");
    code = Tcl_EvalFile(interp, "init.tcl");

    Tcl_DeleteInterp(interp);

    return 0;
}
```


Команды в языке Tcl

1. простые команды: `set`, `for`, `if`, `switch`, ...

```
set a 10
set b "bbb"
set c $a$b
```

2. специальные команды: `#`, `\`, `[`, `]`, `"`, `{`, `}`, ...

- `$` - команда подстановки значения переменной вместо её имени;
- `#` - команда комментария;
- `\` - команда переноса на новую строку;
- `"` и `{ }` – команды группировки символов с подстановкой значений и без подстановки соответственно;
- `[]` – команда выполняет вычисление аргумента внутри скобок и возвращает результат выполнения.

Работа с переменными: простые переменные

Запись в общем виде:

```
set varName ?value?
```

Объявление переменных:

```
set var1 7  
set var2 "This is a string"
```

Использование переменных:

```
set var3 $var2  
# var3 = "This is a string"
```

Bash code:

```
var1=5  
echo $var1
```

```
var2="This is a string"  
echo $var1 $var2
```

Tcl code:

```
set var1 5  
puts $var
```

```
set var2 "This is a string"  
puts $var1 $var2 - ошибка!
```

```
puts "$var1 $var2" - нет ошибки
```

Работа с переменными: составные переменные

Объявление списков (аналог массива):

```
set var4 {1 2 3 4 5}
```

```
set item_2 [lindex $var4 2] <- item_2=3
```

Объявление массивов (ассоциативных массивов) :

```
array set ints {  
    1 2  
    2 9  
    3 100  
    4 50  
}
```

```
set item_3 $ints(3) <- item_3=100
```

Вывод данных на экран

Вывод строки на экран (в общем виде) :

```
puts ?-newline? ?channelid? string
```

Вывод строк на экран (с переводом строк):

```
puts "This is a first string"  
puts "This is a second string"
```

Вывод строк на экран (без перевода строк):

```
puts -newline "This is a first string"  
puts "This is a second string"
```

Вывод с явным указанием канала:

```
puts stdout "This is a string"
```

Чтение данных с клавиатуры

Ввод строки с клавиатуры (в общем виде) :

```
gets channelId ?variable?
```

Считывание с указанием переменной:

```
gets stdin var  
puts $var
```

Считывание без указания переменной:

```
set var [gets stdin]
```

Команда сравнения if-elseif-else (1)

Синтаксис:

```
if expr1 ?then? body1 elseif expr2 ?then? body2 elseif ... ?else? ?bodyN?
```

Пример кода:

```
set x 1
```

```
if {$x == 2} {puts "$x равно 2"} else {puts "$x не равно 2"}
```

```
if {$x != 1} {  
    puts {$x != 1 (не равно)}  
} else {  
    puts {$x равно 1}  
}
```

Команда сравнения if-elseif-else (2)

Пример на if elseif:

```
set x 5
```

```
if {$x == 2} {  
    puts "$x равно 2"  
}  
elseif {$x == 3} {  
    puts "$x равно 3"  
}  
elseif {$x == 4} {  
    puts "$x равно 4"  
}  
elseif {$x == 5} {  
    puts "$x равно 5"  
}  
elseif {$x == 6} {  
    puts "$x равно 6"  
}  
else {  
    puts "Ни одно из рассмотренных"  
}  
}
```

Команда множественного выбора switch

Синтаксис:

```
switch ?options? string pattern body ?pattern body ...?
```

Пример кода:

```
set x 4
```

```
switch $x {  
    1 -  
    3 -  
    5 -  
    7 -  
    9 { puts "Value is odd" }  
    2 -  
    4 -  
    6 -  
    8 { puts "Value is even" }  
    default { puts "Value is greater than 10" }  
}
```


Работа со строками: сравнение (1)

Синтаксис:

```
string compare ?-nocase? ?-length <число>? string1 string2  
string equal   ?-nocase? string1 string 2
```

Пример кода:

```
set x "Hello"  
  
if { [string equal $x "Hello"] == 1 } {  
    puts "$x is Hello"  
} else {  
    puts "$x is not Hello"  
}
```

Пример кода:

```
set x "Hello"  
  
if { [string compare $x "Hello"] == 0 } {  
    puts "$x is Hello"  
} else {  
    puts "$x is not Hello"  
}
```

Прочие команды работы со строками (1)

Получение символа в строке по индексу – команда `string index`

Синтаксис:

```
string index string_val index_val
```

Получение подстроки по диапазону – `string range`

Синтаксис:

```
string range string_val index1 index2
```

Получение длины строки – `string length`

Синтаксис:

```
string length string
```

Циклы в Tcl: циклы for и while

Синтаксис:

```
for start test next body
```

```
for { set x 0 } { $x < 10 } { incr x } {  
    puts "x is $x"  
}
```

Операторы `break`, `continue`:

```
for { set x 0 } { $x < 10 } { incr x } {  
    if { $x == 5 } {  
        continue  
    }  
    puts "x is $x"  
}
```

Синтаксис:

```
while test body
```

Пример:

```
set x 1
```


```
while { $x < 5 } {  
    puts "x is $x"  
    set x [expr { $x + 1 }]  
}
```

Списки в Tcl: объявление (1)

Объявление списков (аналог массива):

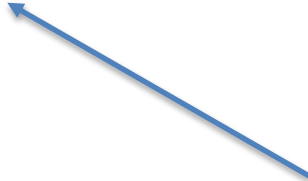
```
set var4 {1 2 3 4 5}
```

Элементы списка - в фигурных
скобках



```
set item_2 [lindex $var4 2] <- item_2=3
```

Работа с элементами списка – только
через команды



Способы объявления списка:

1. **объявлением переменной с указанием элементов;**
2. командой создания списка с указанием отдельных элементов;
3. командой разделения строки на элементы

Списки в Tcl: объявление (2)

Объявление списков (аналог массива):

```
set var4 [list 1 2 3 4 5]
```

```
set item_2 [lindex $var4 2] <- item_2=3
```

Список создаётся командой list

Работа с элементами списка – только
через команды

Способы объявления списка:

1. объявлением переменной;
2. командами создания списка из отдельных элементов;
3. командой разделения строки на элементы

Списки в Tcl: объявление (3)

Объявление списков (аналог массива):

Список создаётся командой split

```
set var4 [split "1,2,3,4,5" ","]
```

```
set item_2 [lindex $var4 2] <- item_2=3
```

Работа с элементами списка – только
через команды

Способы объявления списка:

1. объявлением переменной;
2. командами создания списка из отдельных элементов;
3. **командой разделения строки на элементы**

Команды работы со списками

Объявление списка:

```
set var4 [list 1 2 3 4 5]
```

Добавление элемента в список:

```
lappend var4 6 <- var4={1 2 3 4 5 6}
```

Обращение к элементу списка:

```
set item_2 [lindex $var4 2] <- item_2=3
```

Добавление элемента в список:

```
set var4 [linsert $var4 3 9] <- var4={1 2 3 9 4 5 6}
```

Замена (обновление значения) элемента в списке:

```
set var4 [lreplace $var4 2 2 7] <- var4={1 2 7 9 4 5 6}
```

Определение вхождения элемента в список:

```
lsearch -exact $var4 2 <- 1
```

Одномерные и многомерные списки

Одномерные списки:

```
set list_var1 {1 2 3 4}
```

```
set list_var2 {1 2 3 four five 6}
```

Многомерные списки:

```
set list_var3 {{1 2 3} {4 5 6} 7 8}
```

```
set list_var4 {1 2 {3 4} {} 5 6}
```


Вывод списка на экран

«Топорный» способ:

```
#!/usr/bin/tclsh
```

```
set var { 1 2 3 4 5 6 }
```

```
puts $var
```

```
● student@localhost:~/tcl> ./test.tcl  
  1 2 3 4 5 6  
○ student@localhost:~/tcl> █
```

С использованием цикла for:

```
#!/usr/bin/tclsh
```

```
set var { 1 2 3 4 5 6 }
```

```
for {set i 0} {$i < [llength $var]} {incr i} {  
    puts [lindex $var $i]  
}
```

```
● student@localhost:~/tcl> ./test.tcl  
  1  
  2  
  3  
  4  
  5  
  6  
○ student@localhost:~/tcl> █
```

Обход элементов списка: команда foreach (1)

Синтаксис команды foreach:

```
foreach varName list body
```

```
#!/usr/bin/tclsh
```

```
set var { 1 2 3 4 5 6 }
```

```
foreach i $var {  
    puts "Item = $i"  
}
```

```
#!/usr/bin/tclsh
```

```
set var { 1 2 3 4 5 6 }
```

```
set j 0  
foreach i $var {  
    puts "Item no.$j = $i"  
    incr j  
}
```

Пример работы с двумя индексами:

```
#!/usr/bin/tclsh
```

```
set var { 1 2 3 4 5 6 }
```

```
set k 0  
foreach {i j} $var {  
    puts "Iteration $k:"  
    puts -nonewline "odd = $i; "  
    puts "even = $j"  
    puts ""  
    incr k  
}
```

Работа с аргументами командной строки

За аргументы командной строки в Tcl отвечают три переменные:

1. argc – число передаваемых аргументов;
2. argv0 – имя скрипта;
3. argv – список аргументов.

```
#!/usr/bin/tclsh
```

```
puts "Script name      : $argv0"  
puts "Args count      : $argc"
```

```
set x 0
```

```
while {$x < $argc} {  
    puts "Arg [expr {$x + 1}] : [lindex $argv $x]"  
    incr x  
}
```

Файловый ввод-вывод в Tcl: чтение файлов

```
#!/usr/bin/tclsh
```

```
set fp [open "data.txt" r]  
set file_data [read $fp]  
close $fp
```

```
set data [split $file_data "\n"]
```

```
foreach line $data {  
    puts stdout $line  
}
```

Работа с объектами файловой системы

Проверка существования файла

Синтаксис:

```
file exists name
```

Проверка того, является ли файл файлом, каталогом, является ли файл исполняемым

Синтаксис:

```
file isfile name
```

```
file isdirectory name
```

```
file executable name
```

Извлечение расширения файла

Синтаксис:

```
file extension name
```

Использование подпрограмм

```
#!/usr/bin/tclsh
```

```
proc say_hello {} {  
    puts stdout "Hello from Tcl proc!"  
}
```

```
say_hello
```

```
#!/usr/bin/tclsh
```

```
proc say_hello {} {  
    set x 4  
    puts stdout "Returning 4 from proc!"  
    return $x  
}
```

```
set y [say_hello]  
puts stdout $y
```

```
#!/usr/bin/tclsh
```

```
proc say_hello {name} {  
    puts stdout "Hello, $name, form Tcl proc!"  
}
```

```
say_hello Dima
```

Исполнение внешнего кода

- `exec` – исполнение команд ОС, запуск программ и т.д.
- `eval` – исполнение строки как кода Tcl.
- `source` – загрузка файла и исполнение его, как Tcl скрипта.

```
#!/usr/bin/tclsh
```

```
set var [exec ls -al]  
puts $var
```

Переменная

Код Tcl

```
set cmd { puts "Evaluating puts" }  
puts "CMD IS: $cmd"  
eval $cmd
```