МИЭТ
Национальный исследовательский университет «МИЭТ»
Кафедра ПКИМС

# Компьютерные технологии в научных исследованиях

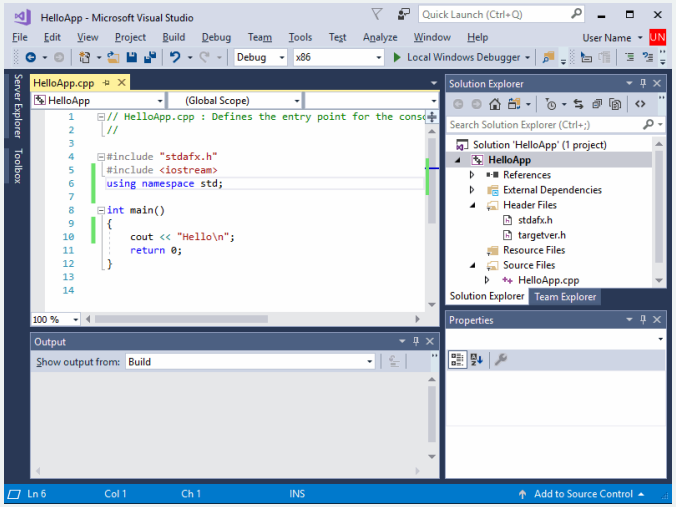

Семинар №1

**Язык Tcl.**

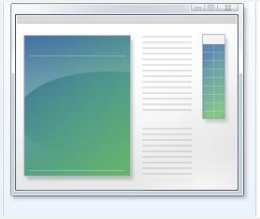
**Основные команды.**

Лекции доступны по адресу:
**http://dima.pkims.ru/courses/5_ktni/**

# Скриптовые языки vs Компилируемые языки (1)

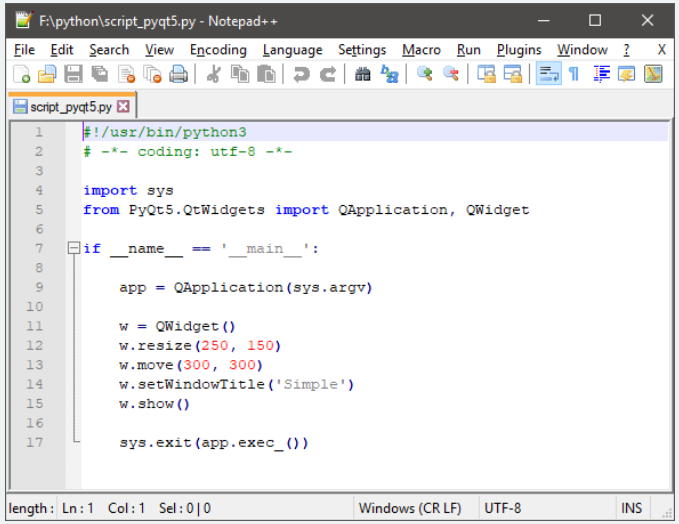



Компиляция и сборка

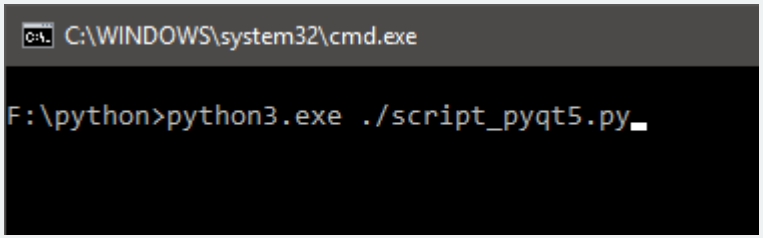


Бинарный исполняемый файл

# Скриптовые языки vs Компилируемые языки (2)




Интерпретатор языка Python



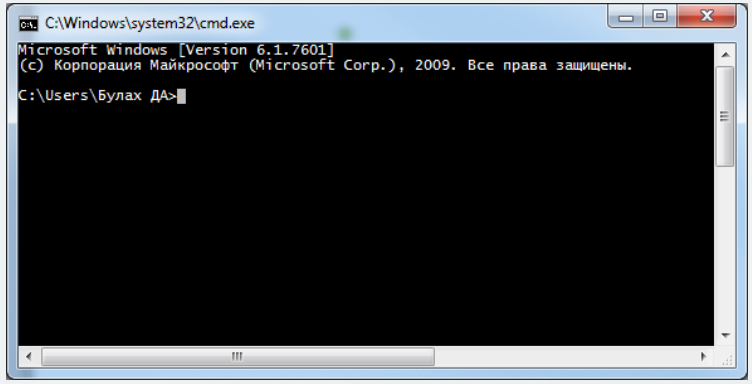

# Характеристика скриптовых языков

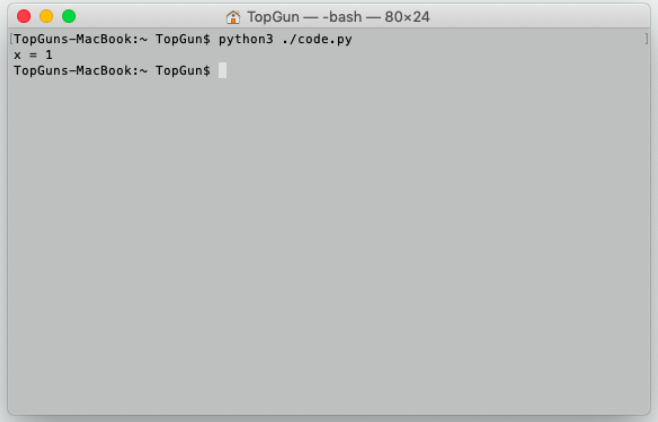
1. Простота языковых конструкций.
2. Высокая скорость написания простых приложений.
3. Поддержка команд операционной системы на уровне языка.
4. Динамическая типизация данных.
5. Переносимость кода между платформами и архитектурами.

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.

# Классификация скриптовых языков

## Языки командных интерпретаторов




## Языки программирования






# Скриптовые языки в САПР

## Обычные языки программирования


python

XILINX
ALL PROGRAMMABLE™

Mentor Graphics®

SYNOPSYS®
Silicon to Software™

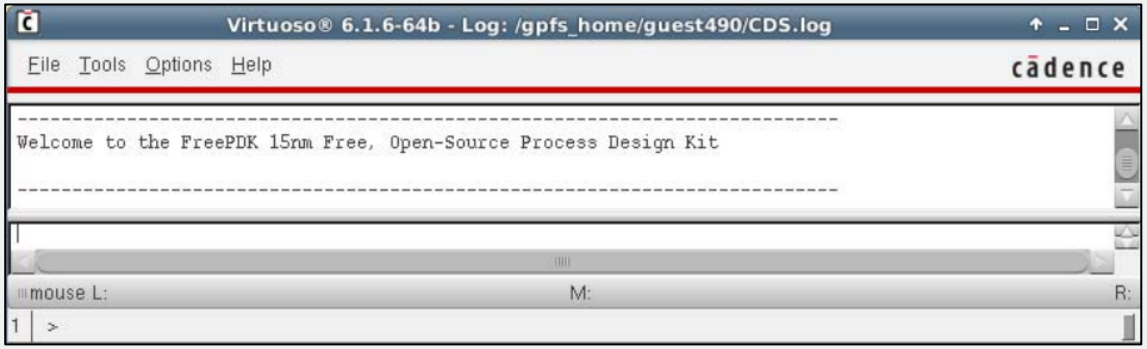
## Свои языки программирования

```
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" )
```

cādence®

# Скриптовые языки в САПР Cadence: OCEAN Script

```
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") )
```

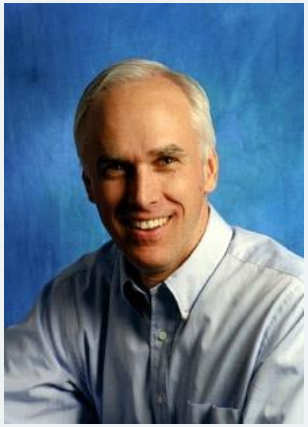


# Язык Tcl (1)

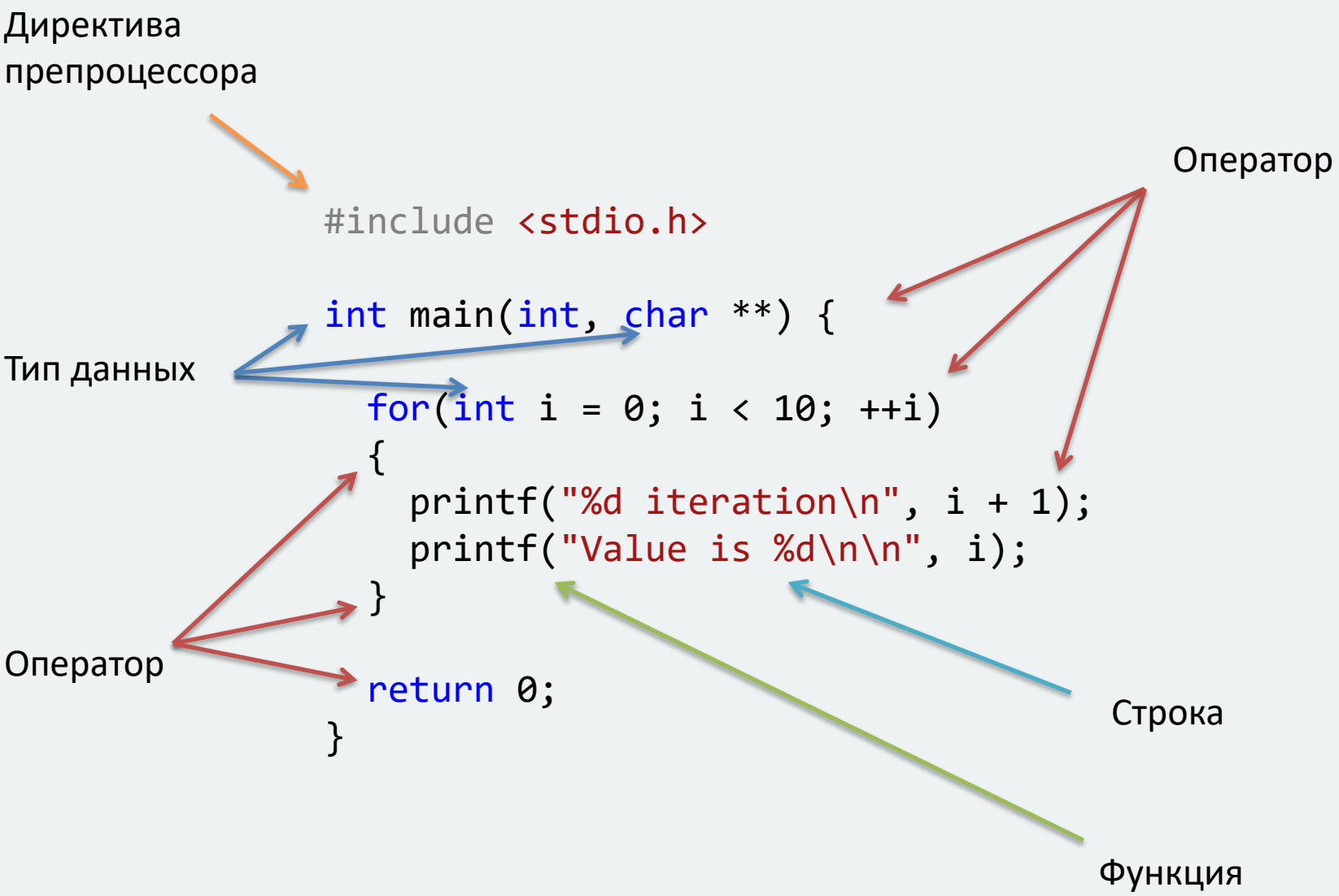
TCL (англ. «Tool Command Language») – один из самых мощных интерпретируемых языков программирования.

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

Разрабатывается с 1988 года

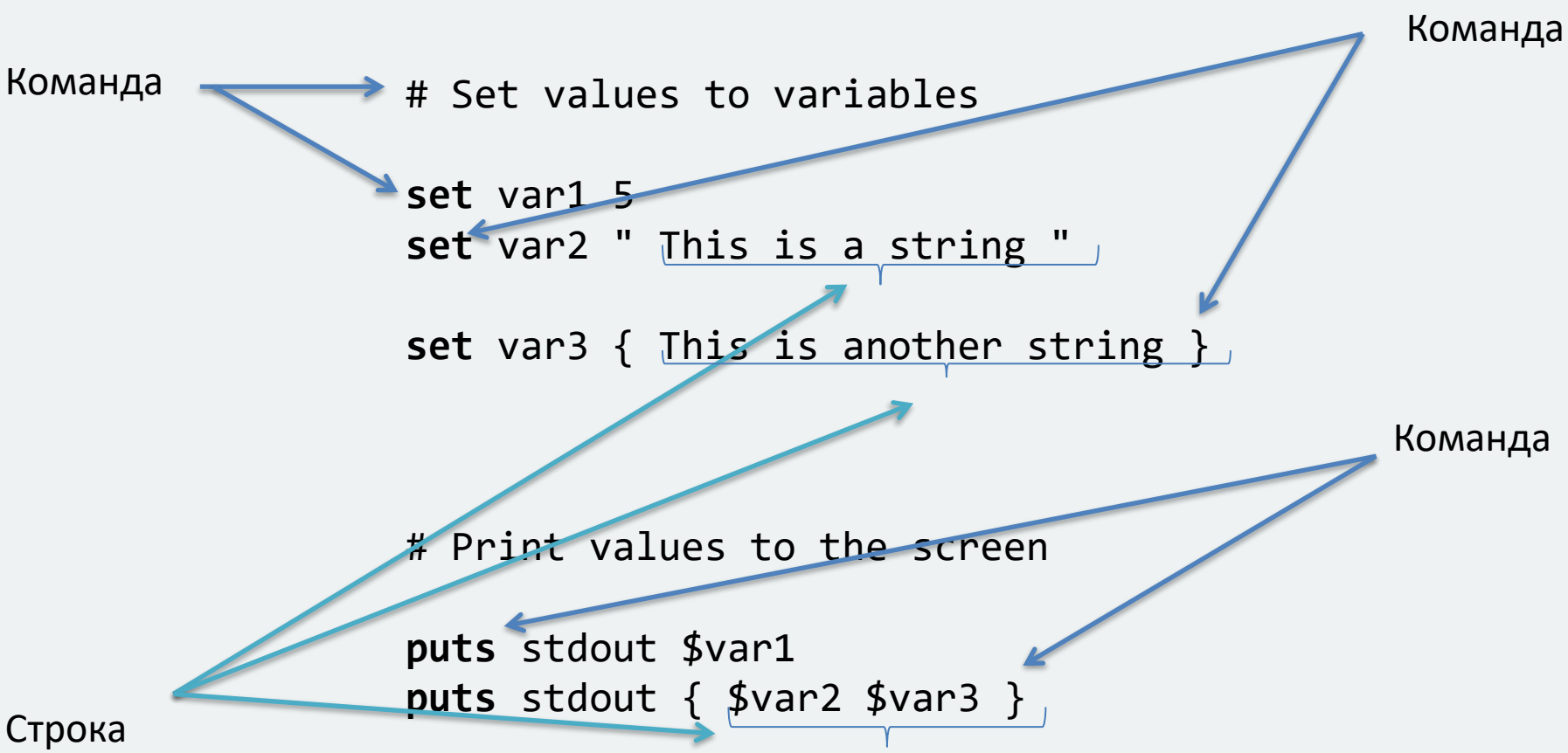


# Структура программ на привычных языках программирования



# Структура программ на языке Tcl

```
#!/usr/bin/tclsh

# Set values to variables

set var1 5
set var2 " This is a string "

set var3 { This is another string }

# Print values to the screen

puts stdout $var1
puts stdout { $var2 $var3 }
```




Изредка встречаются имена переменных, значения, опции команд

# 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. Позволяет описывать процедуры.

```
#!/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]
```

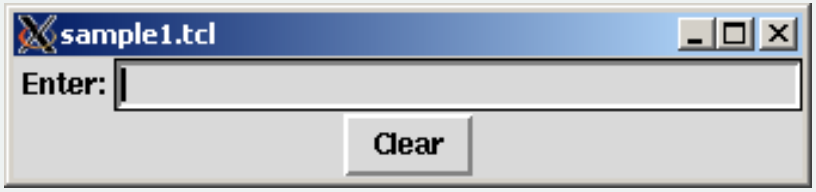
На экран будет выведено:
5

```
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.




# Фишки языка Tcl (3)

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

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

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

using namespace std;

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

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

```
#!/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: #, \, [,],”, {, }, …

# Специальные команды Tcl

- $ - 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"
```

# Операторы группировки { } и " "

Bash code:

```
var1=5
echo $var1

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

Фигурные скобки во многих командах используются как операторные скобки и обязаны начинаться на той же строке, что и оператор, который их использует.

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 ?-nonewline? ?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 –nonewline "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 не равно 2"
} elseif {$x == 4} {
    puts "$x не равно 2"
} elseif {$x == 5} {
    puts "$x не равно 2"
} elseif {$x == 6} {
    puts "$x не равно 2"
} else {
    puts "Ни одно из рассмотренных"
}
```

# Работа со строками в Tcl : сравнение (1)

Синтаксис:

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

Пример кода:

```
set x "Hello"

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

# Работа со строками в Tcl : сравнение (2)

Синтаксис:

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

Пример кода:

```
set x "Hello"

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

# Работа со строками в Tcl : работа 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
```

# Команда множественного выбора switch (1)

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 then 10"}
}
```

# Команда множественного выбора switch (2)

```
switch $x {
  1 { puts "one" }
  2 { puts "two" }
  3 { puts "three" }
  default { puts ">3"}
}
```

```
switch $x \
1 { puts "one"   } \
2 { puts "two"   } \
3 { puts "three" } \
default {
  puts ">3"
}
```

```
switch $x 1 {
  puts "one"
} 2 {
  puts "two"
} 3 {
  puts "three"
} default {
  puts ">3"
}
```

```
switch $x 1 { puts "one" } 2 { puts "two" } ...
```

# Команда цикла for

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"
}
```

# Команда цикла while

Синтаксис:

```
while test body
```

Для этого цикла также можно применять команды break и continue

Пример:

```
set x 1

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

Сколько раз выведется на экран?

```
set x 0

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

# Списки в Tcl: объявление (1)

Объявление списков (аналог массива):

Элементы списка - в фигурных скобках

```
set var4 {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 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):

Список создаётся командой list

```
set var4 [list 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

# 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**

# Списки в Tcl: 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
```

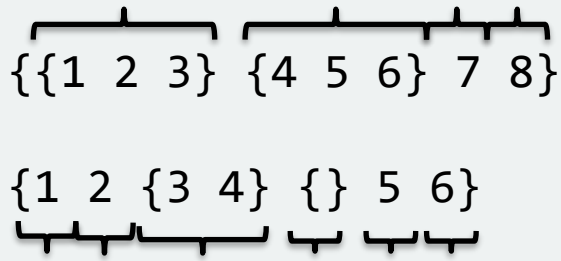
# Одномерные и многомерные списки

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}
```



# Обход списков: команда foreach (1)

Синтаксис команды foreach:

```
foreach varName list body
```

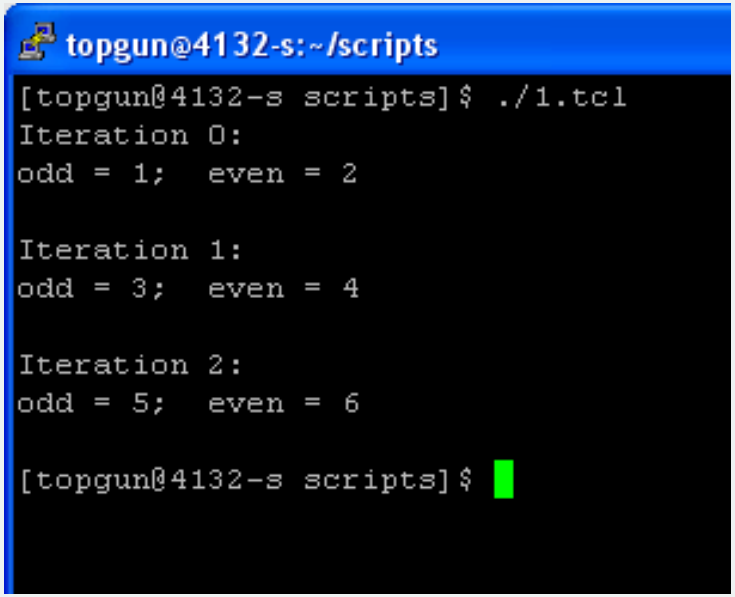
```
#!/usr/bin/tclsh

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
}
```

# Обход списков: команда foreach (2)

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 в Tcl

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 (1)

Дескриптор файла

```
set fp [open "data.txt" r]

set file_data [read $fp]

close $fp
```

Все данные файла

```
set fp [open "data.txt" r]

set file_data [gets $fp]

close $fp
```

Одна строка файла

# Файловый ввод-вывод в Tcl: чтение файлов (2)

```
#!/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

}
```

# Получение информации о файлах

Проверка существования файла
Синтаксис:

```
file exists name
```

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

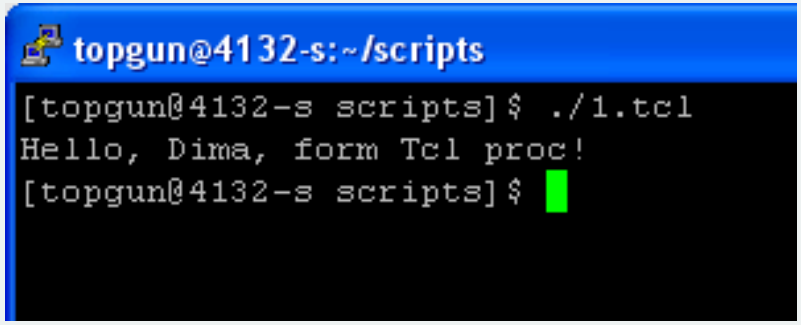
```
file isfile name
file isdirectory name
file executable name
```

Извлечение расширения файла
Синтаксис:

```
file extension name
```

# Использование подпрограмм (1)

```
#!/usr/bin/tclsh

proc say_hello {} {
  puts stdout "Hello form Tcl proc!"
}

say_hello
```




```
#!/usr/bin/tclsh

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

say_hello Dima
```

# Использование подпрограмм (2)

```
#!/usr/bin/tclsh

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

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

# Использование регулярных выражений

```
regexp template string ?matchVar? ?subMatcVar1? …
```

```
regexp {\w+(\d)\w+(\d)} "a1b2" v1 v2 v3
  v1 = "a1b2"
  v2 = "1"
  v3 = "2"
```

```
regexp {^module\s(\w+)} "module inverter (in, out);" a b
  a = "module inverter"
  b = "inverter"
```

# Исполнение внешних программ

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
```