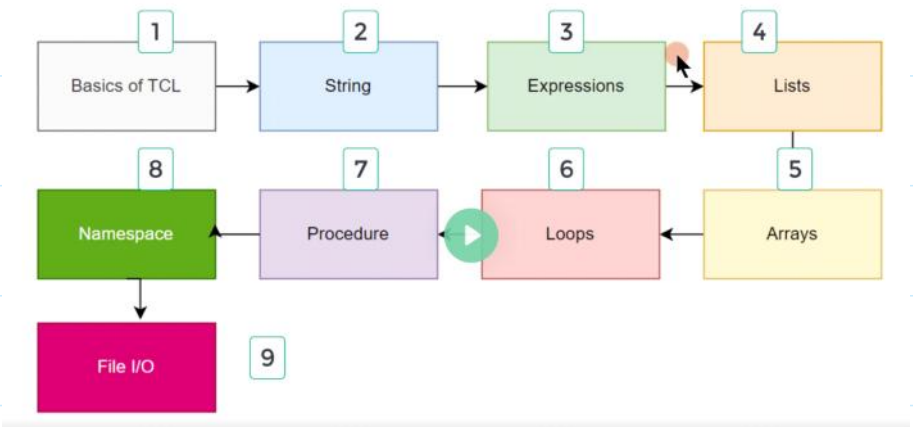
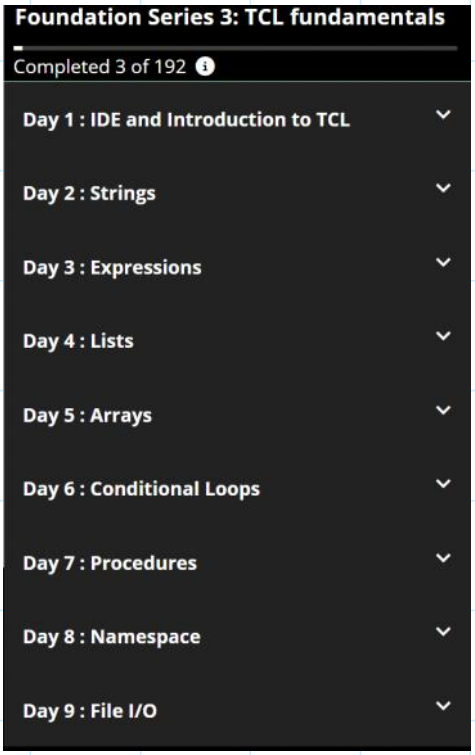
DAY 1: TCL Fundamentals

28 December 2024 11:47





DAY 1

TCL IDE AND BASICS

- How to get TCLKIT
- TCL basics
- TCL License
- TCL Command format
- Set and unset command
- Substitutions : Variable, Command and Backslash
- Variable Naming Convention

- Tcl (Tool Command Language) is a scripting language, a string-based command language, and an interpreted language.
- in Tcl, everything is a string. All data types can be treated as strings.

```
(Downloads) 1 % set var1 12
12
(Downloads) 2 % set var2 34
34
(Downloads) 3 % puts $var1$var2
1234
```

Description

- Tcl is widely used in VLSI design for automating various stages of the design flow, including synthesis, implementation, placement, and routing.
- It enables engineers to customize and streamline design processes to meet specific area, power, and timing requirements.
- Tcl scripts are used to manage multiple files, configure tool settings, and execute repetitive tasks efficiently, making it easier to handle complex designs.

- Additionally, Tcl is essential for performing tasks like Static Timing Analysis (STA), power analysis, and extracting critical timing and power data after implementation.
- Mastery of vendor-specific Tcl commands is crucial for optimizing and automating VLSI design workflows, enhancing productivity and design accuracy.
- The course covers a comprehensive introduction to Tcl over 9 days, starting with basics such as using Tclkit, command formats, and variable handling.
- It progresses to string manipulation, including regexp and regsub, and then explores expressions like arithmetic and logical operations.
- The course delves into lists and arrays, highlighting their differences and commands, followed by conditional loops such as if-else, switch, while, and for
- Procedures, namespaces, and their usage are covered in detail, and finally, the course concludes with file I/O, teaching file handling commands for reading and writing files.

TCL/Tk

- Tool Command Language (Tcl) is a **high-level scripting language**(closer to human language than machine language) commonly used for **automating tasks**, especially in software development, electronic design automation (EDA), and system administration. It's known for its flexibility and simplicity, allowing users to write scripts that control tools, configure environments, or manage data.
- **Simple Syntax:** Tcl has an easy-to-learn syntax, making it accessible even for beginners.
- **Extensible:** Tcl can be extended with additional libraries and modules to add more functionality.
- **Embedding:** Tcl can be embedded within applications, making it popular in tools like Vivado and ModelSim for FPGA design.
- Tcl is often used in environments where rapid development and script-based automation are needed.

- Tool Command Language (Tcl) is a **high-level scripting language**(closer to human language than machine language) commonly used for **automating tasks**, especially in software development, electronic design automation (EDA), and system administration. It's known for its flexibility and simplicity, allowing users to write scripts that control tools, configure environments, or manage data.
- **Simple Syntax:** Tcl has an easy-to-learn syntax, making it accessible even for beginners.
- **Extensible:** Tcl can be extended with additional libraries and modules to add more functionality.
- **Embedding:** Tcl can be embedded within applications, making it popular in tools like Vivado and ModelSim for FPGA design.
- Tcl is often used in environments where rapid development and script-based automation are needed.

- An interpreted language is a type of programming language in which the code is executed line-by-line by an interpreter rather than being **compiled into machine code** all at once before execution.
- Unlike compiled languages (like C or C++), interpreted languages don't require a compilation phase where code is converted to machine code. Instead, the interpreter directly executes the code.
- The same source code can run on different platforms as long as the appropriate interpreter is available.
- Tcl (Tool Command Language) is released under the BSD-style license. This license is permissive, meaning it allows the software to be freely used, modified, and distributed, both in open-source and proprietary applications.
- You can use Tcl in both open-source and commercial projects without many restrictions.
- This makes Tcl an attractive choice for developers who need a scripting language that can be easily integrated into both free and proprietary software.

- Tool Command Language (Tcl) is a **high-level scripting language**(closer to human language than machine language) commonly used for **automating tasks**, especially in software development, electronic design automation (EDA), and system administration. It's known for its flexibility and simplicity, allowing users to write scripts that control tools, configure environments, or manage data.
- **Simple Syntax:** Tcl has an easy-to-learn syntax, making it accessible even for beginners.
- **Extensible:** Tcl can be extended with additional libraries and modules to add more functionality.
- **Embedding:** Tcl can be embedded within applications, making it popular in tools like Vivado and ModelSim for FPGA design.
- Tcl is often used in environments where rapid development and script-based automation are needed.

- An interpreted language is a type of programming language in which the code is executed line-by-line by an interpreter rather than being **compiled into machine code** all at once before execution.
- Unlike compiled languages (like C or C++), interpreted languages don't require a compilation phase where code is converted to machine code. Instead, the interpreter directly executes the code.
- The same source code can run on different platforms as long as the appropriate interpreter is available.

- TCL script contains one or more words.
- Words are separated by spaces or tabs.
- The first word is the command name, and the remaining words are the parameters of the command
- TCL commands are separated by semicolon or newline.
- Command and variables both are case sensitive.
- First word represent TCL command while rest words are arguments for command

```
command_name arg1, arg2;
command_name arg1, arg2
```

```
set fCode "Hello World"
```

set command

- set command is used to assign values to variables.
- The command is followed by two parameters: the first parameter is the variable name; the second parameter is the variable value.
- Although Tcl has no substantial restrictions on variable naming, from the perspective of coding style, it is still recommended that variable names contain only letters, numbers, and underscores to avoid ambiguity caused by variable names.

```
set   variable_name   variable_value ;
```

```
(TCL Scripting) 5 % set var1 1
1
(TCL Scripting) 6 % set var2 2
2
(TCL Scripting) 7 % puts $var1$var2
12
(TCL Scripting) 8 % set fcode "Hello W
rld"
Hello World
(TCL Scripting) 9 % puts "Hello World'
Hello World
(TCL Scripting) 10 % set pi 3.14
3.14
(TCL Scripting) 11 % set root4 2
2
(TCL Scripting) 12 %
```

Summary

Variable declaration is not required in TCL, We need to assigned variable =, no defined variable will throw error.

```
(TCL Scripting) 13 % set vari
can't read "vari": no such variable
(TCL Scripting) 14 %
```

Assigning without a value is not allowed.

- TCL script contains one or more words.
- Words are separated by spaces or tabs.
- The first word is the command name, and the remaining words are the parameters of the command
- TCL commands are separated by semicolon or newline.
- Command and variables both are case sensitive.
- First word represent TCL command while rest words are arguments for command

command_name arg1, arg2;

command_name arg1, arg2

set fCode "Hello World"

unset command

- The function of the unset command is the opposite of the set command.
- This command will cancel the variable definition and release the memory space occupied by the variable.
- it is illegal to cancel an undefined variable

unset variable_name;

- If you need to determine whether a specified variable has been defined, you can use the command

Info exists variable_name;

If the variable exists, it returns 1, otherwise it returns 0

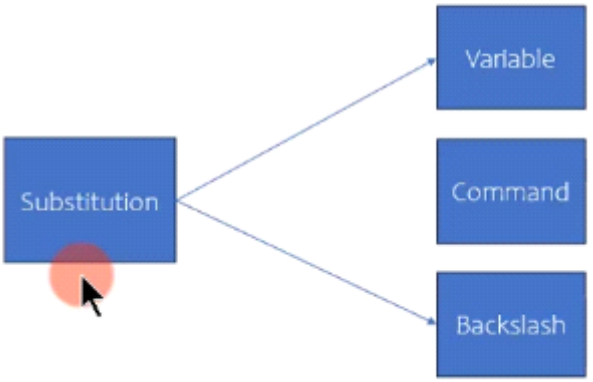
```
(TCL Scripting) 16 % set var1 12
12
(TCL Scripting) 17 % unset var1
(TCL Scripting) 18 %
```

```
(TCL Scripting) 19 % puts $var1
can't read "var1": no such variable
(TCL Scripting) 20 %
```

```
(TCL Scripting) 20 % set var1 12
12
(TCL Scripting) 21 % set var2 13
13
(TCL Scripting) 22 % set var3 14
14
(TCL Scripting) 23 % info exists var4
0
(TCL Scripting) 24 % info exists var3
1
(TCL Scripting) 25 %
```

```
(TCL Scripting) 24 % info exists var3
1
(TCL Scripting) 25 % unset var3
(TCL Scripting) 26 % info exists var3
0
(TCL Scripting) 27 %
```

Substitutions



variable

command

backslash

```
(TCL Scripting) 29 % set var1 3
3
(TCL Scripting) 30 % set var2 var1
var1
(TCL Scripting) 31 %
```

```
(TCL Scripting) 29 % set var1 3
3
(TCL Scripting) 30 % set var2 var1
var1
(TCL Scripting) 31 % set var2 $var1
3
(TCL Scripting) 32 %
```

NAMING CONVENTION FOR VARIABLES:

```
(TCL Scripting) 32 % set var3.1 45
45
(TCL Scripting) 33 % puts $var3.1
can't read "var3": no such variable
(TCL Scripting) 34 %
```

```
(TCL Scripting) 34 % set var3-1 45
45
(TCL Scripting) 35 % puts $var3-1
can't read "var3": no such variable
(TCL Scripting) 36 %
```

- The Tcl interpreter treats "-" and "." as string delimiters, so these two characters should be avoided in variable names as much as possible.
- If these two characters are unavoidable, the variable name must be enclos in "{}" when replacing the variable.

```
(Downloads) 17 % set var.2 12
12
(Downloads) 18 % puts $var.2
can't read "var": no such variable
(Downloads) 19 % puts ${var.2}
12
```

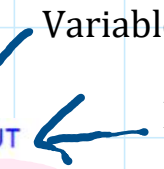
```
(Downloads) 20 % set res1 "LUT"
LUT
(Downloads) 21 % puts $res16
can't read "res16": no such variable
(Downloads) 22 % puts ${res1}6
LUT6
```

```
(TCL Scripting) 36 % set var3_1 67
67
(TCL Scripting) 37 % puts $var3_1
67
(TCL Scripting) 38 % puts $var3_1
67
(TCL Scripting) 39 %
```

```
(TCL Scripting) 40 % puts ${var3.1}
45
(TCL Scripting) 41 % puts ${var3-1}
45
(TCL Scripting) 42 %
```

use {} for 3.1 & 3-1

Appending string to a variable




```
(TCL Scripting) 42 % set res LUT
LUT
(TCL Scripting) 43 % puts $res6
can't read "res6": no such variable
(TCL Scripting) 44 % puts $res 6
can not find channel named "LUT"
(TCL Scripting) 45 % puts ${res 6}
can't read "res 6": no such variable
(TCL Scripting) 46 % puts ${res} 6
can not find channel named "LUT"
(TCL Scripting) 47 % puts ${res}6
LUT6
(TCL Scripting) 48 %
```

include variable name in {}string to be added

incr command

- You can also assign values to variables using the command incr.
- The command is followed by two parameters: the first parameter is the variable name; the second parameter is the variable value.
- If there is no second parameter, the operation of adding 1 to the first parameter is executed.
- If there is a second parameter, the operation of adding the first parameter to the second parameter is executed

```
incr   variable_name   variable_value ;
                       optional
```

```
(TCL Scripting) 48 % incr var2
4
(TCL Scripting) 49 % incr var3 5
5
(TCL Scripting) 50 % incr var3
6
(TCL Scripting) 51 %
```

to declare a variable use a ***set*** method or ***incr*** method

Command substitution

- Command substitution causes all or part of the words of a command to replaced by the result of another command.
- Command substitution is the second substitution form of Tcl, which is expressed in the form of "[]".
- Try to put only one Tcl command in the "[]" for command replacement t enhance the readability of the code and facilitate later code debugging.

```
(Downloads) 23 % set var4 [set var2 12]
12
(Downloads) 24 % puts $var4
12
```

```
(TCL Scripting) 51 % set var1 56
56
(TCL Scripting) 52 % #[], $, \
>
(TCL Scripting) 53 % set var2 [set var1 87]
87
(TCL Scripting) 54 % puts $var2
87
(TCL Scripting) 55 %
```

1

2

type 1 2 3
[] $ \

Backslash substitution

- Similar to C language, the backslash in Tcl is mainly used to insert charac that are considered special symbols by the Tcl interpreter into a word, s as newline, "[", space, "$", etc.

```
(Downloads) 25 % set var1 $5
can't read "5": no such variable
(Downloads) 26 % set var1 \$5
$5
(Downloads) 27 % set ver2 mem[addr]
invalid command name "addr"
(Downloads) 28 % set ver2 "mem[addr]"
invalid command name "addr"
(Downloads) 29 % set ver2 \mem[addr]
invalid command name "addr"
(Downloads) 30 % set ver2 mem\[addr]
mem[addr]
(Downloads) 31 % set var3 \\x
\x
```

$ is used for variable substitution that's why

```
(TCL Scripting) 55 % set var1 $5
can't read "5": no such variable
(TCL Scripting) 56 %
```

```
(TCL Scripting) 55 % set var1 $5
can't read "5": no such variable
(TCL Scripting) 56 % set var1 \$5
$5
(TCL Scripting) 57 %
```

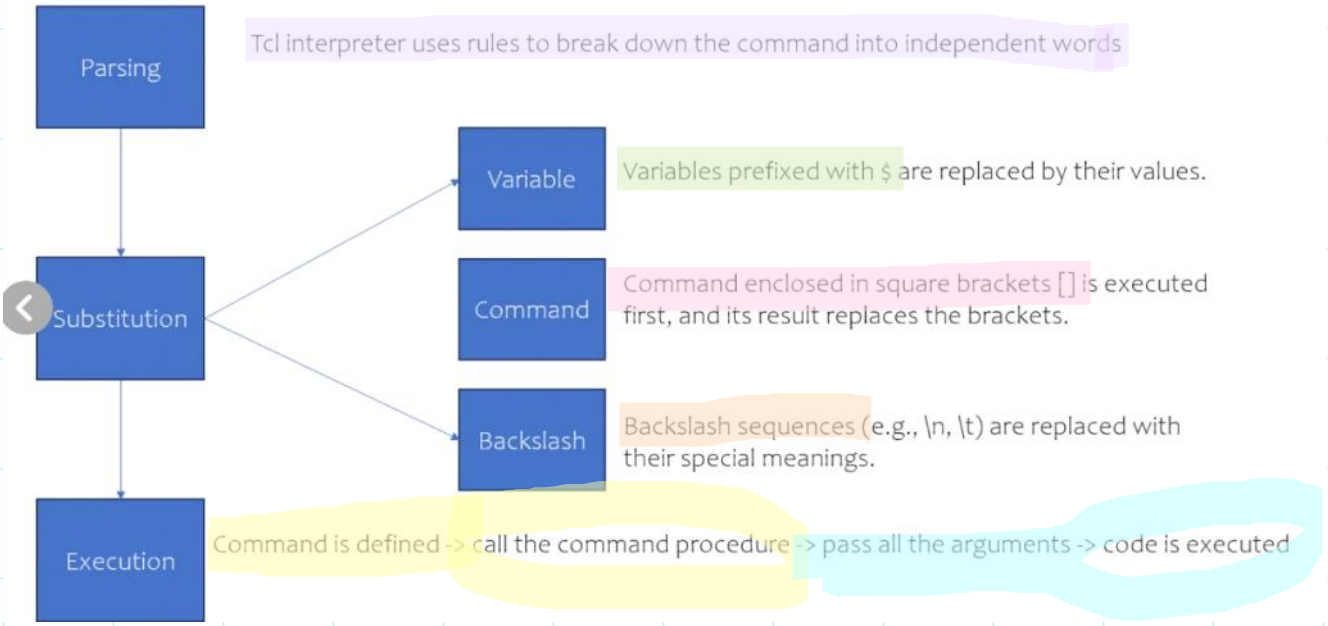
[] is used for variable substitution that's why

```
(TCL Scripting) 57 % set var2 mem[addr]
invalid command name "addr"
(TCL Scripting) 58 %
```

```
(TCL Scripting) 57 % set var2 mem[addr]
invalid command name "addr"
(TCL Scripting) 58 % set var2 mem\[addr]
mem[addr]
(TCL Scripting) 59 %
```

```
(TCL Scripting) 59 % set var3 \\n
\n
(TCL Scripting) 60 %
```

TCL Code execution



Parsing

- The command is split into three words: `set`, `var1`, and `12`.

```
(Downloads) 9 % set var1 12
12
```

set var1 12

Substitution

- The TCL interpreter checks for any variable, command, or backslash substitutions.
- In this command, no substitution is needed because there are no variables (no `$`), nested commands (no `[ ]`), or backslash sequences.

Execution

- The interpreter identifies `set` as the command.
- It sees that there are two arguments:
  - The first argument, `var1`, is the name of the variable.
  - The second argument, `12`, is the value to be assigned to `var1`.