Code No: 156CU

R18

JAWAHARLAL NEHRU TECHNOLOGICAL UNIVERSITY HYDERABAD
B. Tech III Year II Semester Examinations, February/March - 2022
SCRIPTING LANGUAGES
(Common to CSE, IT)

Time: 3 hours **Max. Marks: 75**

Answer any five questions
All questions carry equal marks

- - -

1.a) Explain the Package Management with RUBYGEMS.
b) List and explain the different canvas widget options. [7+8]

2.a) Discuss the Ruby Type System.
b) Illustrate the Embedding of Ruby to Other Languages. [8+7]

3.a) List and explain the Characteristics of Scripting Languages.
b) Using the Perl program explains push and pop in array implementation. [5+10]

4.a) Give a brief note on dirty hands in internet programming.
b) Discuss the pack and unpack, filesystem. [8+7]

5.a) Explain Tk-Visual Tool Kits, Events, and Binding with examples.
b) Why is TK used? Explain TK with an example. [8+7]

6.a) Explain the Tk module in Ruby.
b) How does the jukebox work? Explain. [8+7]

7. List and explain PERL data types with suitable examples. [15]

8.a) How to run a package in tcl, how to remote login to a system A from system B, execute commands in it.
b) Give an example of how to write internet applications using TCL. [8+7]

--ooOoo—

Answer Key

1. a) Explain the Package Management with RUBYGEMS:

1. Installation: RubyGems allows for the easy installation of Ruby libraries (gems). You can install gems using the command `gem install <gem_name>`.
2. Version Management: It supports multiple versions of the same gem, allowing developers to specify which version they need in their projects.
3. Dependency Management: RubyGems manages dependencies between gems, ensuring that the correct versions of dependent gems are also installed.
4. Repository: Gems are typically hosted on the central repository, RubyGems.org, where they can be easily published and accessed.
5. Gemfile: Projects often use a `Gemfile` (with Bundler) to specify the gems needed for the project, making it easy to set up and maintain consistent environments across different setups.

b) List and explain the different canvas widget options:

1. Rectangle: Used to draw rectangular shapes. Attributes like width, height, and fill color can be customized.
2. Oval: Used to draw oval shapes, including circles. It is defined by the bounding box coordinates.
3. Line: Used to draw straight lines. Attributes such as line width, color, and dash patterns can be customized.
4. Text: Used to display text strings at specified coordinates on the canvas. Font, color, and size can be customized.
5. Image: Used to display images (like PNG or JPEG) on the canvas. Images can be positioned, scaled, and manipulated.

2. a) Discuss the Ruby Type System:

1. Dynamic Typing: Ruby is dynamically typed, meaning variables do not have a fixed type and can hold objects of any type.
2. Duck Typing: Ruby relies on duck typing, where an object's suitability is determined by the presence of certain methods and properties rather than the object's type.
3. Classes and Modules: Ruby has a robust class-based system with single inheritance and modules (mix-ins) for sharing functionalities across classes.
4. Primitive Types: Ruby includes a variety of primitive types, such as integers, floats, strings, arrays, hashes, symbols, and more.

5. Type Checking: Type checking in Ruby is typically done at runtime, allowing for greater flexibility but also requiring more comprehensive testing to catch type errors.

b) Illustrate the Embedding of Ruby in Other Languages:
1. Ruby C API: The Ruby interpreter provides a C API, allowing Ruby code to be embedded within C programs. This is useful for performance-critical applications.
2. SWIG: Simplified Wrapper and Interface Generator (SWIG) can generate wrapper code for integrating C/C++ code with Ruby, making it easier to call C/C++ functions from Ruby.
3. Rice: A C++ interface to Ruby's C API, simplifying the process of writing C++ extensions for Ruby.
4. Java (JRuby): JRuby is a Ruby interpreter implemented in Java, allowing Ruby code to be run within Java applications and facilitating interaction between Ruby and Java code.
5. Embedding Examples: Common examples include using Ruby to script game engines, embed business logic within larger systems, or use Ruby for rapid prototyping within applications written in other languages.

3. a) List and explain the Characteristics of Scripting Languages:
1. Interpreted: Scripting languages are typically interpreted rather than compiled, allowing for quick execution and testing of code.
2. High-Level: They provide high-level constructs and are designed to automate frequently used tasks, making them easier to write and understand.
3. Dynamic Typing: Scripting languages often use dynamic typing, where types are determined at runtime.
4. Ease of Use: They are designed for ease of use, often with simpler syntax and powerful built-in functionalities.
5. Integration: Scripting languages are designed to integrate well with other software components, making them ideal for tasks like web development, system administration, and data processing.

b) Using the Perl program, explain push and pop in array implementation:
```perl
#!/usr/bin/perl
use strict;
```

```
use warnings;

my @array = (1, 2, 3);
print "Initial array: @array\n";

# push operation
push(@array, 4);
print "After push: @array\n";

# pop operation
my $last_element = pop(@array);
print "After pop: @array\n";
print "Popped element: $last_element\n";
```

1. Initialization: The array `@array` is initialized with elements (1, 2, 3).
2. Push Operation: The `push` function adds an element (4) to the end of the array.
3. Pop Operation: The `pop` function removes the last element of the array and returns it.
4. Output: The program prints the state of the array before and after the `push` and `pop` operations.
5. Example Run: The script demonstrates adding and removing elements from an array, showcasing how `push` and `pop` work.

4. a) Give a brief note on dirty hands in internet programming:

1. Direct Coding: Writing low-level code for protocols (HTTP, FTP) and network communication, often bypassing high-level libraries.
2. Manual Configuration: Manually setting up and configuring servers, databases, and network settings.
3. Security Considerations: Handling security aspects like encryption, authentication, and dealing with vulnerabilities directly.
4. Error Handling: Dealing with network errors, timeouts, and unreliable connections, often requiring robust error handling and retry mechanisms.
5. Performance Tuning: Optimizing the performance of internet applications, including managing bandwidth, latency, and server load.

b) Discuss the pack and unpack, filesystem:

1. Pack Function: In Perl, `pack` converts a list into a binary representation. It is useful for creating binary data structures.
2. Unpack Function: `unpack` is the reverse of `pack`, converting binary data back into a list. This is useful for reading binary files or network data.
3. Filesystem Operations: In Perl, filesystem operations include reading from and writing to files, handling directories, and managing file permissions.
4. Example of Pack/Unpack: `pack("C*", @array)` converts an array of numbers into a string of bytes. `unpack("C*", $string)` converts a string of bytes back into an array of numbers.
5. File Handling Example:
   ```perl
   open(my $fh, '>', 'file.txt') or die "Could not open file: $!";
   print $fh "Hello, World!";
   close($fh);
   ```

5. a) Explain Tk-Visual Tool Kits, Events, and Binding with examples:
1. Tk Toolkit: Tk is a GUI toolkit for creating graphical user interfaces. It provides various widgets like buttons, labels, and text fields.
2. Widgets: Tk includes a variety of widgets such as `Button`, `Label`, `Entry`, `Text`, and `Canvas` for building GUIs.
3. Events: Tk uses an event-driven model where user actions (like clicks and key presses) trigger events.
4. Binding: Events can be bound to widget actions. For example, binding a button click to a function call.
5. Example:
   ```perl
   use Tk;
   my $mw = MainWindow->new;
   my $button = $mw->Button(-text => "Click Me", -command => sub { print "Button Clicked\n" })->pack;
   MainLoop;
   ```

b) Why is TK used? Explain TK with an example:
1. Cross-Platform: Tk is cross-platform, running on Windows, macOS, and Unix-like systems.

2. Ease of Use: Tk is relatively easy to use, with a straightforward API for creating and managing GUI components.
3. Integration: Tk can be integrated with various programming languages, including Perl, Python, and Ruby.
4. Flexibility: It provides a wide range of widgets and customization options, making it suitable for both simple and complex GUIs.
5. Example:

```perl
use Tk;
my $mw = MainWindow->new;
$mw->Label(-text => 'Hello, World!')->pack;
$mw->Button(-text => 'Exit', -command => sub { exit })->pack;
MainLoop;
```

6. a) Explain the Tk module in Ruby:

1. Installation: The Tk module in Ruby can be installed via the RubyGems package manager with `gem install tk`.
2. MainWindow: The primary window for Tk applications in Ruby is created using `TkRoot`.
3. Widgets: Various widgets such as `TkButton`, `TkLabel`, and `TkEntry` are available for building GUIs.
4. Event Handling: Tk in Ruby supports event handling and binding, allowing developers to link widget actions to functions.
5. Example:

```ruby
require

'tk'
root = TkRoot.new { title "Hello, World!" }
TkLabel.new(root) {
  text 'Hello, World!'
  pack { padx 15; pady 15; side 'left' }
}
Tk.mainloop
```

b) How does the jukebox work? Explain:

1. Song Selection: Users can browse and select songs from a list or library.
2. Play Controls: Typical jukeboxes have controls for play, pause, stop, next, and previous tracks.
3. Queue Management: Users can create and manage a queue of songs to be played in sequence.
4. Playback: The jukebox handles playback of selected songs, often with features like volume control and playback progress.
5. Example: A software jukebox in Ruby might use a combination of Tk for the GUI and a library like `mpg123` for audio playback.

7. List and explain PERL data types with suitable examples

1. Scalars:

Description: Scalars are single data values, which can be a number, a string, or a reference.

Syntax: Scalars are prefixed with a `$`.

Examples:

```perl
my $number = 42;
my $string = "Hello, World!";
my $reference = \$number;
```

2. Arrays:

Description: Arrays are ordered lists of scalars.

Syntax: Arrays are prefixed with an `@`.

Examples:

```perl
my @numbers = (1, 2, 3, 4, 5);
my @strings = ("apple", "banana", "cherry");
print $numbers[0];  # Outputs: 1
```

3. Hashes:

Description: Hashes are collections of key-value pairs.

Syntax: Hashes are prefixed with a `%`.

Examples:

```perl
my %fruit_colors = ('apple' => 'red', 'banana' => 'yellow');
```

```
print $fruit_colors{'apple'};  # Outputs: red
```

4. Typeglobs:
Description: Typeglobs are used to create symbol table entries and can store references to all types of data.
Syntax: Typeglobs use `*`.
Examples:

```perl
*x = \$y;
```

5. References:
Description: References are scalar values that hold the location of another value.
Syntax: References are created using a backslash (`\`).
Examples:

```perl
my $array_ref = \@numbers;
my $hash_ref = \%fruit_colors;
```

6. Filehandles:
Description: Filehandles are used for file input and output operations.
Syntax: Typically use uppercase names.
Examples:

```perl
open(my $fh, '<', 'file.txt') or die "Cannot open file: $!";
while (my $line = <$fh>) {
    print $line;
}
close($fh);
```

7. Special Variables:
Description: Perl has a range of special variables that have predefined functions.
Syntax: Special variables often use punctuation symbols.
Examples:

```perl
print "Current line number: $.";
```

8. Undefined Values (undef):
Description: `undef` is used to indicate undefined values or to reset a variable.
Syntax: Using `undef`.
Examples:

```perl
my $undefined = undef;
```

9. Context-sensitive Values:
Description: Perl variables can behave differently in scalar and list context.
Syntax: Implicit context changes.
Examples:

```perl
my @array = (1, 2, 3);
my $count = @array;  # Scalar context: $count is 3
```

10. Complex Data Structures:
Description: Perl allows complex data structures using references to create multi-dimensional arrays, hashes of arrays, etc.
Syntax: Use of references.
Examples:

```perl
my @AoA = (
    [1, 2, 3],
    [4, 5, 6],
    [7, 8, 9],
);
print $AoA[1][2];  # Outputs: 6
```

8. a) How to run a package in TCL, how to remote login to a system A from system B, execute commands in it:
1. Running a Package: In TCL, packages are loaded using the `package require` command. For example, `package requires http`.
2. Remote Login: Use `ssh` for remote login. For example, `exec ssh user@systemA`.
3. Executing Commands: Commands can be executed remotely using `ssh` and TCL's `exec` command. For example, `exec ssh user@systemA "ls -l"`.
4. Example:

```tcl
package require Tcl
exec ssh user@systemA "ls -l"
```

5. Automation: TCL scripts can automate the process of logging in and executing multiple commands.

b) Give an example of how to write internet applications using TCL:

1. HTTP Package: TCL's `http` package can be used to create internet applications. It provides functions for making HTTP requests.
2. Socket Programming: TCL supports socket programming, enabling communication over TCP/IP.
3. Web Server: Simple web servers can be created using TCL for handling HTTP requests and responses.
4. Example:

```tcl
package require http
set url "http://example.com"
set token [http::geturl $url]
set response [http::data $token]
puts $response
http::cleanup $token
```

5. Web Applications: TCL can be used to develop web applications with frameworks like TclHttpd, providing server-side scripting and templating.