Unit-1

Solutions

Introduction to PERL and Scripting

1.1 Scripts and Programs

Q1. Write briefly about scripts and programs.

Answer :

Scripts

A script is a set of instructions that are interrupted. Scripts are written using a different kind of language called scripting language. Perl, Phython, Bash etc., are some of the examples of scripting languages. Scripts are created and modified by the end-users. Scripting languages have the following features,

1. They build applications from already available software components.
2. They control applications that have a programmable interface.
3. They give importance to faster development of program rather than the run-time efficiency of the program.

Program

A program is a set of instructions that are compiled. Programs are written using programming languages like C, C++, Java etc. The action of writing these programs using conventional languages is called programming. Programs are created and modified by the programmers. They run faster than the scripts as they are compiled and not interrupted. Programming languages have the following features,

1. They develop large applications from scratch.
2. They employ only professional programmers.
3. They start building applications from well-defined specifications and meet the specified performance constraints.

Scripting languages are different from conventional programming languages. Because the kind of programming that they are used for is qualitatively different from conventional programming. The most important difference is that scripting language enhances the productivity of the user by allowing to write programs even though they are not programmers by profession.

1.2 Origin of Scripting

Q2. Discuss about origin of scripting

Answer :

Origin of Scripting

The term 'script' was introduced in early 1970s. In Unix World, a sequence of commands that are read from a file were executed in sequence as if they were typed in at the keyboard. The originators of UNIX operating system termed this file as a 'shell script'. The popularity and usage of scripting language got increasingly more powerful in UNIX world. It extended to other language processors such as 'Awk script', 'Perl script' etc. Thus a script is nothing but a text file containing a sequence of commands that is executed directly rather than being compiled before execution.

Here we discuss two occurrences of the term 'scripts'.

The first occurrence was in DOS-based system. To establish a dial-up connection to a remote system, a communication package is required. This package wrote scripts using a proprietary language to automate the sequence of operations involved in establishing the connection. This usage is much similar to the film and television scripts, which contain a sequence of detailed instructions for the camera operators and the actors. Here the usage of script caused the modem to perform certain actions.

The second occurrence was in hyper text system called the Apple Macintosh Hyper-card application. The uses was allowed to define sequence of actions associated with mouse clicks and movements using hypertalk language. They were called scripts rather than programs. Thus, makings all the non-programmers comfortable. Here the script was used to make changes in some aspects of the display the programming concepts such as loops and branches add value to scripts and make them interesting.

1.3 Scripting Today

Q3. What are the three different meanings of the term "scripting"?

Answer :

The term 'scripting' has three different meanings which are discussed below,

1. Scripting is a new style of programming which is developed to run applications much faster than traditional methods. This style of programming uses scripting language. It also sees that all the applications meet the changing user requirements. Applications which are built by interconnecting all the available components (that are written in conventional language) are called glue applications and the scripting language is called glue language.

Example: To build graphical user interfaces using visual basic, the visual controls which are already build are used.

2. ECMA script controls applications that provide programmable interface (may be an API). It manipulates, customizes and automates the facilities of an existing system using scripting language. Generally, an application is build from a collection of objects. The properties and methods of these objects are related to scripting language.

   Examples: To control MS office applications we use visual basic.

   To create interactive and rich webpages we use client scripting and dynamic HTML.

3. Scripting language is used as an alternative to conventional language. Its rich functionality is used for programming; particularly for system programming and for system administration.

   Example: To handle system maintenance tasks and the administrators of windows NT systems. UNIX system administrators have been adopting the scripting language, Perl for a long time.

1.4 Characteristics of Scripting Languages

Q4. Discuss the various characteristics of a scripting language.

Answer : Model Paper-I, Q1(a)

Characteristics of Scripting Language

Different scripting languages have many features in common. Some of them are characterized below,

1. Compiled and Run Together

This feature is very important. Scripting languages are sometimes described as interpreted languages. Though they behave as if they are interpreted, describing them as interpreted language seems to be an over implication. Scripting languages are executed immediately without performing compilation and run separately. The concept of immediate execution serves better because scripting is often as interactive activity. The cycle of editing, compiling, linking and then running the program (like in conventional language) is not suitable for interactive applications.

There are few scripting languages which strictly act as interpreters. The source file is read in single forward pass without any lookhead or backtracking. When a valid keyword or construct is recognized then appropriate operations are performed. Now a days, many languages are employing the hybrid technique i.e., compiling to an intermediate form and then interpreting. This intermediate form is represented in the form of a parse tree. This technique is widely used because the source code is available at run-time and it also provides messages in case of error.

2. Minimum Overhead and Easy to Use

Scripting languages are easy to use as the declaration of variable is optional. When the variables are used, they are declared and initialized to some meaningful value. There are limited data types and data structures. The default data type is string. The string can be converted to number when required. Most often used data structures are associative arrays the conventional indexed arrays are rarely used. There is no limit on the size and shape of arrays.

3. Increased Functionality

The functionality of scripting languages have been increased in few areas. Based on the use of regular expressions many languages provide string manipulation, while very few languages provide access to low-level OS or to the object model or to the API easily.

4. Less Concentration on Efficiency

Scripting languages are easy to use at the cost of efficiency. Scripts build using scripting languages are mostly used only once. The rest of the scripts are modified to new the requirements of new application. Thus, performance and efficiency is less concentrated and more concentration is on how fast the application is build.

1.5 Uses for Scripting Languages

Q5. Explain the traditional applications of scripting languages.

Answer :

Applications of traditional scripting languages are as follows:

1. System Administration

The requirements of system administration in the unix world lead to the introduction of the concept 'scripting'. To perform the daily work like adding new uses to the system or making backup copies of file system etc., the administrators used the shell scripts. They executed a sequence of shell commands from a particular file.

The Bourne shell can be claimed to be one of the first scripting languages that can be used to obtain sophisticated effects with shell scripts. It is best suited to system administration because of the following features,

(i) Bourne shell is fullys programmable

(ii) It provides complete access to the underlying system calls of the operating system.

Apart from the general requirements, there are some other requirements for system administration. They are,

(i) Processing the contents of the files.

(ii) Displaying the results in a form that is readable

(iii) Tabulating the disk usage etc.

These requirements needs extreme text processing since shell scripts are not best suited for these requirements. There should be a language that supports all the activities such as providing easy access to underlying system calls, handling textual data etc. In addition, the language developed should have low overhead.

Awk is a scripting language specialized in handling textual data. However the AWK scripts need to be embedded in shell scripts to perform system administration activities. This creates problem since syntaxes of AWK and shell scripts are not compatible. Perl solves these problems by containing the capabilities of both the shell scripts and AWK and adds its own powerful facilities.

2. Experimental Programming

To write large and complex programs, we require different tools and a team of programmers. Some of the traditional languages like 'C' provide tools for this purpose. Apart from the complex programs, there is another set of programs that involve experimentation because exact requirements are not known at the beginning of programming. And there is often need to produce several versions of the program before producing the final one.

Thus scripting languages with low overhead are best suited for experimental programmings. Examples of such programming is perl, the support of perl for experimental programming spread its usage in rapid application development and building prototypes.

This language provides modern application consisting of programmable API with scripting language.

Example: To manipulate the editing and formating engine of microsoft word, use the extension language 'wordbasic'.

3. Command-line Interfaces

Another application of scripting languages is to combine together different parts of code from various languages through the command-line interface. It allows to create new applications by combining various parts of other applications. Thus scripting language is used as a glue. In unix world, glue allows a program to run another program get the results to send it to third program as input. In fine form of pipes, the shell provides basic form of glue. The complete capabilities of 'glue' is provided by perl language.

The development of a large software system must use a combination of two language. One language such as C, C++ or Java should manipulate the complete internal data structures systems. For this, the system developed must be a combination of two languages like C and a scripting language for writing scripts to combine together different pieces of codes.

4. Controlling Applications

An application of scripting language is to control a dial-up connection established between the remotely situated computers. It is necessary to use scripting language because there are certain situations where in conditional logic is involved.

For instance, whenever a user dials a number, he/she has to wait for the uses on the remote system to answer the call. Upon receiving the response the caller has to acknowledge the response and wait for the 'login' prompt. Once the user enters the user identifier, he/she again has to wait for the 'password': prompt. As it is possible that any of these operations may fail, it is therefore necessary to include conditional logic so as to handle the failures.

An application that has an interactive interface must be automated. An application is said to be scriptable application if it has a control interface that can be controlled or used by another program. Such scriptable application can use application automation. A control interface may be as simple as accepting just a string as command or as complex as the windows API. Remote control languages are often referred to as batch languages in the operating system world.

Q6. Explain the modern applications of scripting languages.

Answer :

Applications of modern scripting are as follows:

1. Visual Scripting

A collection of visual objects or controls that have properties is used to construct a graphical interface. This process of constructing a graphical interface is known as visual scripting. The properties of visual objects include text on button, background and foreground colours etc., these

properties of objects can be changed by writing a program in a suitable language. External events like mouse click, key press etc., make some of these objects (like buttons) to respond. This response to action is defined by the script. Scripting the objects is noting but programming such graphical interfaces.

The outstanding visual scripting system is Visual Basic. It is used to develop new applications. It provides a graphical interface to legacy applications by providing prototypes or final product of their own right. Thus, the developed, prototypes need to replaced by production versions later. Visual scripting is also used to create enhanced web pages.

2. Scripting Components

In the modern world of scripting languages, we use the idea of glue to control the scriptable objects (collection of objects) belonging to the scripting architecture. Microsoft's visual basic and excel are the first applications that used the concept of scriptable objects. To support all the applications of microsoft the concept of scriptable objects was developed.

The following are the scriptable objects.

1. The *'component objects'* that are well built and which encapsulates the functionality and the data of entire application or specific part of an application.

   Example: The spell checker in a word processor application.

2. The visual objects which have attributes that are controlled by a scripting language. These objects are embedded in an application to allow dynamic integration or they become a part of an application like visual basic.

3. *Element* in an object model that describe the structure of entities which are of user interest.

   Example: A web page

4. *Components* of a compound document.

   Example: A spreadsheet in a word-processor document.

An application is constructed by using several available 'component objects'. These component objects from different applications are taken to create the 'compound documents'. These documents are controlled by the 'component ware'. The objects whose attributes (properties) can be set, encapsulate data and functionality (or methods).

The activity of manipulating the interworking of components and objects is exclusively defined by the term 'scripting'. Any program can control the components which is capable of invoking the methods of objects. Using conventional languages (Example: C++).

3. Client/server Web Scripting

For answer refer Unit-I, Q7.

1.6 Web Scripting

Q7. Explain in detail about web scripting

Answer :

Now-a-days, web has become one of the most fertile areas for the applications of scripting languages.

Web scripting is classified into three areas,

1. Processing forms
2. Dynamic web pages
3. Dynamically generating HTML.

1. Processing Forms

In early days, a limited form of user interaction is provided by HTM through the forms. In original web implementation, the information entered in the form by the user is encoded and send to the server for processing, by a CGI script. The server generates an HTML page and sends back to the web browser. String manipulation and text manipulation are required to form processing string manipulation will decode the form data and text manipulation will construct the HTML page. The form processing may also require other system administrative tasks such as establishing network connections etc. Perl is one of the language that is best suited for CG1 scripting.

There is an alternative to form processing in which the form data is validated at the client side within web browser using the scripting language before sending the form to the server. The scripting languages such as Java Script/Jscript, which is available in both Internet explorer and nestscape navigator and VBscript, which is available only in Internet Explorer.

2. Dynamic Web Pages

Dynamic HTML which is implemented only in internet explore translates the components of web page into scriptable objects. This provides dynamic control over the web pages and a simple interface to the user. The scripts for this is written using JavaScript/JScript or VB script languages which are interpreted by the browser. Scripts also provide the information about the web browser hence help in optimization of web pages.

A technology could active X implemented only in internet explorer creates web pages allowing dynamic user interaction by using active-x controls which are the embedded visual objects. These controls are the scriptable objects which can be scripted in many languages. It is not necessary for the scripting host i.e., active *x* to interface directly to an embedded scripting language, rather, it can acquire any one of the scripting engines to run a script specific to the particular engine. The internet explores can even scripted in Perl using Perl scripting engine.

3. Dynamically Generated HTML

A dynamic web page is often created by generating HTML whose scripts are executed on the server. This technique helps in creating web pages whose data can be retrieved from the database.

Example: IIS web server implements ASP in JScript or VBScript.

1.7 The Universe of Scripting Languages

Q8. Explain the universe of scripting languages.

Answer :

Universe of Scripting Language

The world of scripting language consists of traditional scripting and modern scripting. The universe of scripting language consists of many such worlds overlapping with each other. It includes,

- ✓ Unix world
- ✓ Microsoft world
- ✓ VBA world
- ✓ Web scripting world.

The Unix world (used for traditional scripting) is now available on most of the platforms. The microsoft world includes the visual basic and the active X controls. The VBA world is for scripting documents such as compound documents. The web scripting world is for scripting at client-side and at the server-side.

It is found that overlapping these worlds with each other is very complex.

Example using VB script, Java script/JScript, perl or TCL for web scripting.

Recently, the universe of scripting has been enlarged in size with the introduction of world of peel and TCL. They are used in large organizations for implementing complex applications. TCL world is used in banking system development where as leading aeroscape company uses perl to implement an enterprise-wide document management system.

Now a days, due to enormous increase in processor power, the performance hit of interpretation is not much considered than the compilation. By using scripting language, the order of magnitude increased due to gain in productivity will be settled to a larger extend and the universe looks interesting.

1.8 PERL – Names and Values

Q9. What is Perl? What are its uses?

Answer :

Perl

Perl stands for Practical Extraction and Report Language, devised by Larry Wall, primarily for trivial administrative tasks. After its inception in 1987, it has been developed over the years, into a dynamic programming language operating at the server-side. It has seen a steady growth from the first version Perl.0, to the current one i.e., Perl 5.0.

Perl is a programming language, which is neither completely compiled, nor completely interpreted language. Rather, its implementation lies between the two. Perl programs are compiled, not only to check for errors but also to make the run time performance impressive, even though it is interpreted. The *Perl* is a Perl language processing system, which compiles and interprets Perl programs. It also includes a debugger.

Perl finds its use in many ways, like from extracting information from a text-file, conversion of a text-file into another form, making Web pages move user friendly and dynamic etc.

Applications of Perl can be broadly classified into three categories. They are,

(i) Report Writing

Since, it was actually made for report-writing tasks, Perl does it very efficiently.

(ii) System-administration

Once the potential of Perl was realized, it was developed to come handy in system administration, where Unix and Linux are used. Since, the configuration files of both Linux and Unix are in text, which can be manipulated easily using Perl.

(iii) Web-page Creation

Perl is always in contention and considered for creating web pages, that are interactive and needs a small database to store some information.

The other common uses of Perl are, creation of feedback forms, guest book, message boards, homepages, counters etc.

Q10. Explain with an example the execution of a simple Perl program.

Answer :

Programming using Perl is very simple. It adds very low overhead to the simple programs.

Let us consider a simple program to print Hello in Perl. It is written as,

```
print "Hello sure \n";
```

Perl does not use brackets but accepts the following variations,

```
print("Hello sure \n");
print("Hello sure \n");
print("Hello sure", "\n");
print("Hello", " ", "sure", "\n");
```

A Perl program is stored in a file with an extension .pl. For example, the above program is stored in myfile .pl. Now, to run this file and print the statement present in it we use the following command.

```
$ perl myfile.p1
```

As, Perl is an immediate execution language, no separate commands are needed to compile and run.

Execution of Perl Program in UNIX Environment

A Perl script is made an executable command by placing it in an executable file whose first line contains

```
# !/usr/bin/per1
```

This line causes the invocation of the Perl interpreter on rest of the script.

For example, consider a Perl script namely "greeting" to print Q "Hello sure" message.

```
# !/Usr/bin/per1
print "Hello world! \n".
```

The Perl interpreter discards the first line by treating as a comment and executes the next print line.

A Perl script at the UNIX shell prompt is executed as follows,

```
$ greeting
```

Q11. Explain about names and values in Perl.

Answer :

In Perl language, a variable is identified with a name and a value is assigned to it. This can be written as,

```
Variable_name = value;
```

Names

A variable can store a single item of data (a scalar value) or a collection of data items (an array or hash).

In perl language, a variable name starts with a special character that represents the data context i.e., $ for scalar data, @ for arrays, % for hash and & for subroutine. After the initial special character, the variable name always starts with a letter or an underscore followed by one or more letters, digits and underscores. The following are the valid variable names.

Example

```
$var
%a
&b
@list
```

A single non-alphanumeric character can also be used to assign a variable name like $ $, $? such variable name are reserved for the Perl system.

The special character at the beginning of a variable name not only defines the data content but also tells us about the kind of data.

Value

A variable does not require any declaration before its use, it comes to existence when it is used for the first time in the program. The type of the variable depends upon the value assigned to the variable.

Example

```
$ var = 10;  //It explicitly decreases the var as integer variable.
$ char = 'A';  //It is defined as character variable.
```

If a variable is assigned a special value undef. as

```
$ var = undef;
```

and the var variable is assigned to some other variable as shown below,

```
$ xyz = $ var;
```

Then in numeric context, the undef value will be set to zero whereas in string context it is set to an empty string.

Q12. Discuss about scalar data, numeric constants and string constants.

Answer :

Scalar Data

Perl language provides two types of scalar data,

(i) String

(ii) Number.

(i) String

String is a sequence of characters that are stored in available memory as sequence of bytes of unlimited length.

(ii) Number

A number is a number, there is no difference between the integer and real numbers. Numbers are stored as a signed integer or as double length floating point number in the native format of the system. As Perl is a dynamically typed language, the system will take care about the variable whether it is a string or a number. The system performs type conversions implicitly, if necessary.

Scalar data name begins with a dollar sign ($) whether it is predefined or user defined.

The dollar sign is followed by a letter, which can be followed by digits or underscore or any number of letters.

For boolean values Perl uses numberic zero "O" and the empty string to denote false and all other values to denote true.

Numeric Constants

Numeric constants (or numeric literals) can be represented in various ways like octal, hexadecimal, scientific rotation etc. Commas or spaces cannot be used to breakup a large numeric value instead, we use underscores.

Some valid numeric constants are,

642971.131,

642_971_131,

0477 (octal),

ox4ff (Hexa),

125.45 e–5,

16.25 E–2,

String Constants

A string constants (or string literals) is the representation of a single value.

String Literals

A String literal, is the representation of a string value, within the source code of a computer program. Normally, string literal expression begins and ends with a double quotes. The elements or characters that constitute the string are specified from left to right. Perl has two types of string literals.

(i) Single-quoted string literals

(ii) Double-quoted string literals.

Any sequence of characters that begin and end with a single quote is called, 'single-quoted string literal'. While, any character sequence beginning and ending with a double quotes is called double-quoted string literal.

Example

'This is a single-quoted string'.

"This is a double-quoted string".

The single quotes are not part of 'a string'. Instead, they tell Perl, where a string is starting and where it is ending. While, double-quoted string literal is very much part of the string. The major difference between the two is that the characters which cannot be easily entered into the source code using the keyboard, can be entered using double-quoted string literal through its special characteristic called 'Escape sequence'.

Double-quoted string literal, also allows us to insert double quotes inside a string using backslash.

Example

"Tom said, \"learning Perl is real fun;\" ".

The only escape character that can be entered using single-quoted string literal is, the 'backslash'. It is done in two ways either write two backslashes in a row or a backslash followed by a single-quote.

i.e., \\ or \'.

Q13. What is the purpose of the following Perl operators,

(a) q

(b) qq

(c) x

(d) Period.

Answer :

(a) q

This operator is used to specify different delimiter for string literals, with same characteristics of single quoted strings.

Example

q$ we are comfort here, we need to go out$

(b) qq

This operator is used to specify a different delimiters for string literals, with same characteristics of double-quoted strings.

Example

qq@ "yesterday! we went to picnic"@

(c) x

x is used as a repetition operator. Its left operand is a string and right operand is a numeric value.

This operator repeats the string for the number of times which is equal to the right operand (numeric value).

Example

"Happy" x 2

"Happy Happy"

Here, left operand is a string (Happy), right operand is a numeric value (2) and x is the repetition operator.

(d) Period

String concatenation can be done, with the help of an operator called period. It denoted as '**.**'

Example

Let the value of the variable $addition is "plus" the expression $addition. "doctor" has the value "plus doctor".

1.9 Variables

Q14. Write about variables and assignments in Perl.

Answer :

As in C, Perl also uses '=' as the assignment operator. It permits the statements like.

(i) $x = ($y = 9) + 10;

This expression assigns the value 9 to $y, which is added to 10 and the resultant value 19 is then assigned to $x.

(ii) The assignment statement also interpolates the value of a scalar variable enclosed within double quotes.

```
$y = 'latha';
$z = "sri $y";
```

This expression will assign the string $z as "srilatha" with a space in the middle.

(iii) If interpolation of a variable is required without any space between scalar variable and string, then it should be written as follows.

```
$y = "latha";
$z = "sri${y}";
```

Here Braces acts as the delimiter that determines the characters belonging to the variable name.

❖ <STDIN> : <STDIN> is a special variable that can be used where a scalar value is required. It takes a string containing the newline from the standard input that is the keyboard and evaluates the string if there is no value given as input then the perl will wait until a string with a newline is typed and returned.

<STDIN> returns undef when End_of_file is reached. In this case, it evaluates the string as an empty string, which is treated as false in Boolean context.

❖ If <STDIN> is assigned to the right hand side of a scalar variable, the variable present on the left side will be assigned with the input line string.

For example, <STDIN> can be used in while loop as shown below

```
while(<STDIN>) {
-----
-----
-----
}
```

<STDIN> can also be used in an assignment statement. When it appears on the right-hand side of an assignment statement, the string containing the newline is assigned to the scalar variable present on the left-hand side.

The string will be assigned to the anonymous variable if it appears in any other scalar context. The value of this anonymous variable is accessed using $.

1.10 Scalar Expressions

Q15. Explain the various operators available in Perl. Also discuss their associativity and precedence.

Answer :

Expressions are formed by combining the scalar data variables or literals using operators. Perl has many operators that are ranked in 22 precedence levels. All the available operators are categorized into 6 groups.

1. Arithmetic operators
2. String operators
3. Bitwise operators
4. Conditional operators
5. Comparsion (or relational) operators
6. Logical (or Boolean) operators.

1. Arithmetic Operators

Arithmetic or numeric operators in Perl are similar to those in other programming languages. They are,

(i) Binary operators

(ii) Unary operators.

Binary Operators

Operators that operate on two operands are called, 'binary operators'. They are,

Addition (+)
Subtraction (–)
Multiplication (*)
Division (/)
Exponentiation (**)
Modulus (%)

Example: $a = 10 + 20;

(i) Unary Operators

Operators that operate on one operand are called, 'unary operators'. They are,

Increment (++)
decrement (––)
Addition (+)
Subtraction (–)

Example

```
$x = 100;
$y = ++ $x;
```

Now $y contains |0|

2. String Operator

In perl, most of the string processing is done by using built in functions and regular expressions. Perl uses two operators *x* and *a* period.

(i) Perl uses x operator to replicate the string

Example: $a = "Hai" x3

Now, the x operator set the $a as "HaiHaiHai"

(ii) Unlike other programming languages, perl uses a period (.) operator to concatenate two strings.

Example

```
$a = 123
$b = 678
print $a.$b      //result is 123678
print $a + $b    //result is 801.
```

Some operators like auto increment and unary minus can also be used in string context.

(i) Auto-increment

If a variable is assigned with a string of a letters and digits. Then, in string mode, the auto increment operation is applied on the string starting from the rightmost character.

Example

```
$b = 'bo';
print ++$b;      /* prints b1 */
$x = 'AZ9'
print ++$x;      /* prints Bao*/
```

(ii) Unary Minus

When unary minus is applied to a string containing plus or minus as its first character then it returns the same string with the opposite sign.

When a unary minus is applied to a string variable then it returns a string prefixed with a minus.

Example:

Consider A string variable named $ name with a value "sia". If unary minus operator is applied on this string i.e., – $ Name then, it evaluates the strings as "– sia". Unary minus is most useful in creating command strings to be sent to a UNIX shell where the command arguments begin with "—".

3. Bitwise Operators

The unary tilde (~) operator performs a bitwise negation on its numerical operand by generating its 1's complement.

The operators ∧ (exclusive OR) (OR) and (and) are other bitwise operators used in Perl as well as in other languages. These operators are evaluated as follows:

(i) If one of the operands is a number or a variable and has only ever been used as an integer then both operands are converted to integers and bitwise operation is performed on two integers.

(ii) If both operands are strings and they have previously been used as strings only then bitwise operation is performed on the bits of two strings. If one of strings is shorter then it is padded with zeros.

4. Conditional Expression

Conditional expression will chose one value from the two alternatives value at runtime depending on the outcome of a test.

Example: $x = ($a < 1)? 0 : 1;

In this example, if '*a*' is less than 1 then it returns 0, otherwise returns 1. This returned value is assigned to $x.

5. Relational or Comparison Operators

Every programming language has an operator, which is used to define a relationship between any two entities. Such operators are called as 'relational operators'. Perl also defines two separate classes of relational operators, one for string operands and another for numeric operands both of which can take any scalar value.

Relational operators used by Perl, are given below in the table. All the first six relational operators output either +1 (if the entities compared are true) or " " when the comparison fails. While the last relational operator gives '–1', when the right operand is greater than the left operand. It gives '0' when both are equal and '+1' when the left operand is greater than the right operand.

Numeric Operators	String Operators	Operation
==	eq	Is equal to
!=	ne	Is not equal to
<	lt	Is less than
>	gt	Is greater than
<=	le	Is less than or equal to
>=	ge	Is greater than or equal to
<=>	cmp	Compare, returning –1, 0 or +1

Table: Relational Operators

While working with relational operators, there can be little risk involved. This comes on occasion when the system does an inappropriate comparison and does not intimate the user. This instance happens in two cases.

Case 1

When the system is forced to compare string operands using a numeric relational operator, instead of a string relational operator and vice-versa often the result in such cases is an inappropriate comparison and thus result in a wrong output.

Case 2

In discrete cases, system cannot convert certain given operands. It gets hypothetical and produces zero.

Example

(5 < 10) returns true

('5' lt '10') returns false because in the canonical sort order for ASCII strings 10 comes before 5.

6. Boolean or Logical Operators

Apart from relational operators, Perl uses other operators called 'boolean operators' to base its statement control flow. Even this has two sets of same semantic operators but with different precedence.

The operators &&(AND), ||(OR) and !(NOT) are at higher precedence level, while the other operators 'and', 'or' and 'not' are at lower precedence level. When running a program, system will evaluate lower precedence operators, only after all the other operators are evaluated.

Example

Print "Hello" if

$x < 5 and

$y > 10;

This will print Hello when the value of x is less than 5 and y is greater than 10.

Precedence and Associativity

When two operators that have different levels of precedence appear in an expression, which operator should be evaluated first is specified by the precedence rules.

When two operators that have same level of precedence appeal in an expression, then associative rules specify which operator should be evaluated first.

All Perl operators and their associativity are tabulated below in precedence order from highest to lowest.

Operator	Associativity
List operators (leftward)	Left
– >	Left
++, – –	Non-associative
**	Right
!, ~, \, unary +, unary –	Right
= ~, !~	Left
*, /, %, x	Left
+, –, ..	Left
<<, >>	Left
Named unary operators	Non-associative
<, >, <=, >, +, It, gt, le, ge	Non-associative
= =, !=, <=>, eq, ne, cmp	Non-associative
&	Left
|, ^	Left
&&	Left
||	Left
. .	Non-associative
? : '	Right
=, + =, –=, etc.	Right
comma, =>	Left
List operators (rightward)	Non-associative
not	Right
and	Left
or, xor	Left

The unary increment and decrement operator, cannot appear in the same expression separated only by an operand. Hence their associativity is non-associative.

Associativity right, specifies the expression should be evaluated from right to left and left indicates the expression evaluates from left to right.

1.11 Control Structures

Q16. What are the different types of control statements available in Perl? Explain them with an example.

Answer :

Control statements are the statements, that provide a programmer with the luxury of controlling the flow of his program. Statements like 'if', 'elsif', 'else', 'unless' and all the loop statements are the few control statements available in Perl.

All the control statements in Perl, use a block formed by putting braces ({ }) around a sequence of statements, that should be controlled. If block is not used, then it generates an error.

'if', 'elsif' and 'else' Statements

All these statements execute the code only when the given condition is satisfied.

Syntax

```
if(condition)
{
    //code to be evaluated
    //if the condition is true
    }
elsif(condition)
{
    //code to be evaluated
    //if the condition is true
    }
else
{
    //code to be evaluated
    //if the condition are false
}
```

Note: elsif is not elseif.

Example

```
if($name<1)
{
    print("tom \n");
}
elsif($name<2)
{
    print("sawyer \n");
}
else
{
    print("no name \n");
}
```

unless Statement

This control statement is reverse of the *if* statement. It executes the block of statements if the condition is false.

Syntax

```
unless(condition)
{
//code to be executed
}
```

Example

```
unless($value = = 50)
{
    print("$value is greater than or less than 50");
}
```

Loop Statements

Loop statements continuously execute a block of statements, as long as the condition is true. The system will only exit the loop statement, when the condition becomes false. There are three kinds of loop statements,

(i) while loop

(ii) until loop

(iii) for loop.

(i) while Loop

while loop is similar to that used in C.

Syntax

```
while(condition)
{
    loop body statement(s)
}
```

Example

```
$numvalue = 10;
while($numvalue < 12)
{
print("number: $numvalue \n");
    $numvalue++;
}
```

Output

Number: 10

Number: 11

(ii) until Loops

until loop works in a similar fashion as the **while** statement except that, **while** statement executes the block of code as long as the given condition is true, whereas, **until** will execute the block of code, as long as the condition is false. It will exit the loop, once the condition becomes true.

Syntax

```
until(condition)
{
    loop body statement(s)
}
```

Example

```
$value = 1;
until($value > = 4)
{
    print("$value \n");
    $value++;
}
```

Output

1

2

3

(iii) for Loop

for loop statement used in Perl, is quite similar to that used in C. It is used to execute a block as long as the condition is true.

Syntax

```
for(initial expression; conditional expression; increment expression)
        {       loop body statement(s);
        }
```

Example

```
for($i = 0; $i ≤ 5; $i++)
{
        print "$i \n";
}
```

The above code will print values from 0 to 5.

Last, Next and Redo Commands

Last, next and redo are the loop control operators provided by perl to overside the loop normal behaviour, next and last commands are used instead of 'continue' and 'break' respectively these are always present inside a loop and are used to exit the loop, or transfer the control outside the loop.

next

next statement is used, to stop the current iteration of the loop and start the next iteration after checking the condition.

Example

```
$num = 0;
while($num ≤ 5)
{
        if($num == 2)
        {
        print "now next statement will run \n";
                next;
        }
                print $num;
        }
        continue            //continue block continues
                            //the next iteration of the loop
        {
                $num++;
        }
```

Output

```
0
1
Now next statement will run
3
4
5
```

last

It is used to end the loop and transfer the control outside the loop immediately, even if the continue block is also used.

Example

```
$num = 0;
while($num ≤ 5)
{
if($num == 3)
{
        last;
}
        print $num;
}
continue
{
        $num++;
}
```

Output

```
0
1
2
```

It will not print the remaining values.

redo

The redo command restarts the current iteration of the loop without evaluating the condition. If continue block is present then it will not be executed then

Example

```
$a = 1;
$b = 3;
while(1)
{
        $a++;
        $c=4;
        if($ a< 5);
        redo;
}
print("$a $b $c \n");
```

Output

```
5
3
4
```

Q17. What are built-in functions in Perl?

Answer :

Perl categorizes the built-in functions into five groups.

(i) Numeric function
(ii) Scalar conversion functions
(iii) Structure conversion function
(iv) String function
(v) Input/output.

Unlike other languages, Perl treats built-in functions as unary operators that are identified by a name instead of an ideagraph. That is, the arguments of a function should be enclosed in brackets only when there is need to emphasize the precedence of the operator.

A built-in function that takes a list of arguments is called list operator.

A function that takes a single argument is called named unary operators. The precedence rules for unary operator are as follows:

1. If the arguments of a function are enclosed in brackets then the function name (or operator) and its arguments will have highest precedence. Because the operator behaves as a function call.

   Example

   ```
   $x = rand ($y*z) +1;
   ```

2. The precedence level of a named unary operator is lower than arithmetic operators and higher than logical operators. Thus the following two statements have the same effect.

   ```
   $
   x = rand     $y*2 + 1;
   $ x = rand ($y*z + 1);
   ```

   In perl a function name with or without parenthesis will represent the same.

3. If a list operator does not have any brackets then it has very high precedence to the left and very low precedence to the right.

   Example

   ```
   print "sure", "team", "\n";
   ```

   In the above list operator commas bind the elements tighter to result in desired effect.

   When a list operator is used as an element of a list then evaluation is done as follows:

- ❖ Commas an the right of the list operator are evaluated before the operator itself.
- ❖ Commas on the left of the list operator are evaluated after the operator.

Thus ("color", "shapes", substr $str, 2 4) and ("color", "shapes", substr ($str, 2, 4)) are equivalent.

1.12 Arrays

Q18. What is an array? Explain about creating, accessing and processing of arrays.

Answer :

Array

An Array can be defined as a variable, that holds multiple values of same type. Objects of array are called elements, which can be either numeric or string and can also be any scalar type.

1. Creating an Array

In Perl, an array name begins with an 'at' sign (@). Perl assigns a value to array variables, using array assignment operator, which is an equal sign i.e., '='.

Examples

- ❖ @list = (1, 2, 3)

  The list array gets a three element literal.

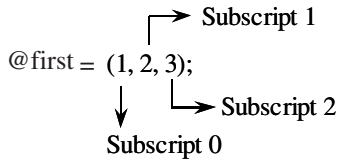
- ❖ @first = ('cat', 'rat');

  @second = @first

The above statement copies contents of @first to @second.

2. Accessing Arrays

In Perl, access to an array element is done, through a numeric index. Every element of an array is tagged with a subscript value, starting from '0' and then increases with an increase in array elements.



Example

```
@first = (1, 2, 3, 4, 5, 6);
$second = $first [3];
```

The above statement accesses the 4th element of @first i.e., numeric 4 and assigns it to the variable $second.

Note

While accessing an array, the symbol '@' for an array name, becomes '$' name in the syntax.

Another Example

```
@first = (1, 2, 3, 4, 5);
$second = $first[1]; #it gives 2 to $second
$first[2] ++; #increments the 3rd element of @first.
($first[0], $first[1]) = ($first[1], $first[0]);
```

It swaps the first two elements of @first, those are the few of many ways to access an array.

3. Array Processing

Array elements are processed using 'foreach' statement.

In Perl, the value of scalar variable is local in a 'foreach' construct. It means the effect of 'foreach' statement will be local and will be limited to its construct itself and will not traverse to other parts of the program.

Example

```
foreach $first(@second) {
$first/=4; #divides every element of @second with 4
and places in $first
}
```

The effect of the above 'foreach' statement will be limited till the flower brackets, beyond it $first will take its original values.

The other characteristic of 'foreach' statement is that, while processing an array it treats vacant places as " ".

Example

@first = (2, , 6, 8, , 12);

@first

2		6	8		12

```
foreach $list(@first) {
print "Next:$list \n";
}
```

Therefore the output will be,

```
next: 2
next:
next: 6
next: 8
next:
next: 12
```

Q19. Write a Perl program to illustrate the use of arrays.

Answer :

The following program illustrate the use of arrays,

```
{
#Initialize an array with name countries
@countries = ("India", "Pakistan");
print "These are the contents of array \n";
print "@countries";
print "<br/>";
#Lets add some content to already present array
push(@countries, "Srilanka");
unshift(@countries, "Bangladesh");
print "These are the new contests of array \n");
print "@countries";
}
```

Output

"These are the contents of array India, Pakistan."

"These are the new contents of array Bangladesh, India, Pakistan, Srilanka".

Q20. In what ways Perl arrays are different from the arrays of other common high-level programming languages?

Answer :

Though the execution and evaluation of arrays in Perl is no different from the other high-level programming languages, but the characteristics that array posses in Perl makes it to standout and mark the difference.

Perl has made arrays more dynamic and flexible in nature, by making it possible to alter its length at any given point of time so much, so that it can even change its length at the time of program execution.

While working with arrays in C, C++ and Java, we are restricted to use similar type of data the arrays. A single array cannot have both numeric and string data inserted into it. In Perl, we can have a single array consisting of both numeric and string data, thus, extricating a programmer from defining different arrays for different scalar data.

An array in Perl can have bigger size, but contain fewer elements. It can have absent elements. Suppose, an array @list contains two elements ('boy', 'girl') whose subscripts are 5 and 12. In this case, it is understood that @list has some absent elements. This feature is absent, in many higher level programming languages.

In C, when an array name without subscripts is assigned to a variable, then that variables is treated as pointer to an array. In contrast, Perl does not treat array name without subscripts as pointers.

Q21. Explain built-in array functions.

Answer :

Built-in Array Functions

Unlike the other high-level programming languages, arrays in Perl can be defined as either operators or functions, Perl chooses to treat them as operators if the parameters are parenthesized and as functions if they are not parenthesis.

New elements are inserted/extracted into/from an array, using four functions '**shift**', '**unshift**', '**pop**' and '**push**'. Shift and push perform the insertion operation but at different ends of the array. Similarly, unshift and pop, performs the extraction operation, but at different ends of the array. If we want to extract a first element from an array, say @second and place it in a scalar or list called, '$first'. Then for this purpose, shift function is used.

Example

```
@num = (1, 2, 3);
$ele = shift@num;
Print "Top element is $ele";
```

Output

Top element is 1

shift function always deals with the left end of arrays. After the execution of a shift function, subscripts of other elements in an array always get reduced by 1.

pop function also does a similar kind of job but from the right end of an array. Execution of pop function does not have any effect on the subscripts of the other array elements. Its job is to remove the last element and return it.

Example

```
@num = (1, 2, 3);
$j = pop@num;
print $j, "\n";
```

Output

"1"

unshift function too works on the left end of an array and its execution increases the subscript value of the array elements. It is used to append a list, or a scalar variable to an array on the lower subscript end i.e., at the beginning of an array.

Example

Suppose, we have an array @first = (10, 11) and we want to add a list (5, 7) on the left end of array like,

```
@first = (10, 11);
unshift @first, (5, 7);
```

then, contents of array @list will be, (5, 7, 10, 11)

Similarly, push function too increases the size of array like unshift but in right direction.

For above example, push function will be used as,

```
@first = (10, 11);
push @first, (5, 7);
```

now, contents of @first will be (10, 11, 5, 7)

1.13 List

Q22. Write about list and scalar context.

Answer :

Perl performs any operation in one of the two contexts. Scalar context or list context.

In scalar context, the target of an operation is a single data item. A $ symbol before a variable establishes a scalar context. For example,

```
$x = 20;
$str = "Ashraf";
```

In list context, the target of an operation is a collection of data items. The @ symbol before a variable establishes a list context. For example,

```
@names = ("Ashraf", "Shazia", "Shamsia", "Asfia");
@numbers = (100, 200, 300);
```

A list can have only one scalar value, thus making it a list of one element. Thus the two statements,

```
@org = "sure";
```

and `@org = ("sure");`

are equivalent.

Some of the important features of a list are give below,

1. When a list is assigned to a scalar value then a list is evaluated in a scalar context assigning the last element of the list to a scalar value. For example,

   ```
   $num = (10, 20, 30);
   ```

   assigns the value 30 to $num.

2. When an array is evaluated in a scalar context then the length of the array is assigned to the scalar value.

   For example,

   ```
   $len = @numbers
   ```

   assigns the value 3 to $len.

3. A function (e.g: print) that expects a list of arguments can also establish a list context.

   ```
   $len = @numbers
   print "length of array is $len"
   ```

   Here print statement displays "length of array is 3".

   However, it may display unexpected results when the same thing is written as,

   ```
   print "length of array is @number"
   ```

   Now,

   Print statement displays length of array is 100 200 300.

   That is print displays the entire contents of the array.

   The desired effect can be obtained by casting the array as shown below.

   ```
   print "length of the array is scalar @ numbers";
   ```

Q23. What are lists? Explain.

Answer : Model Paper-I, Q1(b)

List

A list is a collection of elements like variables, constants or expressions that are separated by commas.

Arrays store lists, so list elements are said to be array elements.

The elements in the list are often enclosed in round brackets, however, this is not a part of list syntax.

Example

"Kavitha", "Malathi", "Niharika"

("A", "B", "C")

(1, 2, 3, 42)

($ a, $b, $c)

($a + $b, $b – $c, $c + $a, $b + $c)

Brackers are only used for grouping purpose that satisfies the precedence rules and it makes the script readable.

Lists allow shorthand notations like,

(1 ..... 9)

("A" ..... "J", "K" ..... "Q", "R" ..... "Z")

A list can use the qw (quote words) operator to quote the strings in a list.

For example,

("Sure", "Spectrum", "Universal")

Can be written using qw as,

qw(sure spectrum universal)

or

qw | sure spectrum universal |

or

qw / sure spectrum universal /

Another feature of lists is that a list containing only variables can be used in an assignment statement as the target and /or as the value to be assigned. Thus, it makes simultaneous assignments as well as swapping of elements without using a temporary variable easy.

Example

($r, $g, $b) = ("Red", "Green", "Blue");

($x, $y) = ($y, $x);

Q24. Explain different methods of iterating over lists.

Answer :

In Perl, there are many ways to perform the same operation on all items in a list. They are as follows:

(a) foreach

(b) map

(c) grep.

(a) foreach

In Perl, the value of scalar variable is local in a 'foreach' construct. It means the effect of 'foreach' statement will be local and will be limited to its construct itself and will not traverse to other parts of the program.

Example

```
foreach $first(@second) {
$first/=4; #divides every element of @second with
4 and places in $first
}
```

The effect of the above 'foreach' statement will be limited till the flower brackets, beyond it $first will take its original values.

If there is a need to convert an array into a list, Perl automatically does this conversion.

The other characteristic of 'foreach' statement is that, while processing an array it treats vacant places as " ".

Example

@first = (2, , 6, 8, 12);

@first

2		6	8		12

```
foreach @list(@first) {
print "Next:$list \n";
}
```

Therefore the output will be,

next: 2

next:

next: 6

next: 8

next:

next: 12

(b) map

map is is an in-built function provided by Perl. This function creates a new list from the existing list by over a list of items. This function has two forms,

Syntax: map expression, list;

map block list;

For each element in the list, the map function evaluates the expression or block and returns a new list.

For example, consider a list of items

@list = ('pen', 'paper', 'pencil', 'eraser', 'scale')

To make the plural forms of above items, we need to add 's' at the end of each word. This can be done by using map function as follows,

@list 1 = map $ – . 's', @list;

Here, $_ is set to the list element. Now the list 1 becomes,

list 1 = ('pens', 'papers', 'pencils', 'erasers', 'scales')

(c) grep

grep function is used to print the elements in a list that match the given pattern. This function takes a list and a pattern as input.

Syntax: grep pattern list

Example

@list = ('chandu', 'chandra', 'chandrashekar', 'chandhru', 'kavishekar')

grep/chand/@list

The 'grep' function returns a new list containing all the elements of the original list that match the pattern 'chand'. The above grep function returns (chandu, chandra, chandrashekar, chandhru)

1.14 Hashes

Q25. Explain in detail about hash variables. What is the difference between hashes and arrays?

Answer :

Hashes

Arrays in which each data element is paired with a key are called, 'associative arrays'. To find a specific data element in an associative array, we use a 'hash' function. Hence, associative arrays in Perl are called 'hashes'.

Percent (%) sign is added before a hash variable, which differentiates if from other variables.

Hash variables can be initialized, using list literals. We can assign arrays to hashes, vice-versa. In this process, a copy is generated.

For example, if hashes is assigned to array, then the subsequent change in hash does not cause a change in array.

In order to associate a key and its data element we use symbol "=>".

Example

%salary = ("Supriya" => 3800, "Madhuri" => 4000);

Accessing Hash Element

Referencing of a hash element is done, using subscripting operation, which uses braces and subscripts, the hash name with a key. Since, the values of a hash are scalar values, the reference to a hash element begins with a dollar sign.

Example

$salary = $salary {"Supriya"};

Insertion Operation

A new element is added to the hash, by assigning a new value to a hash reference indexed with a new key.

Example

$salary {"Swathi"} = 9000;

Deletion Operation

We use delete operator, to remove an element from a hash.

Example

delete $salary {"Madhuri"};

Setting Hash to Empty

There are two ways to set a hash to empty. They are,

1. By assigning empty list to hash.
   Example
   %salary=( );
2. By using undef operator.
   Example
   undef %salary;

Searching An Element

We use *exists* operator, to know whether an element with specific key is in hash or not.

Example

```
If (exists $salary {"Swathi"})
    #some code
```

Assigning Hashes to Array

The operators *keys* and *values* are used, to extract the keys and values of a hash into an array.

Example

```
foreach $emp(keys %salary)
{
    print "salary is $salary {$emp} \n";
}
@salaries = values %salary;
print "All the salaries: @salaries \n";
```

Difference between Hashes and Arrays

The two main differences between hash and arrays are,

1. To find data elements, arrays use numeric subscripts whereas, hashes use string values (keys).
2. In arrays, the order of elements depends on index. In hashes, it depends on the internal hash function, used for inserting and accessing hash elements.
3. An array name begins with an **at** @sign whereas, a hash name begins with a **percent** (%)sign

Q26. Discuss the following hash operators,
(a) Delete
(b) Reverse.

Answer :

(a) Delete

The *delete* function is used to delete the hash elements.

Syntax: delete $hash {$ key};

This function takes a reference to a hash element and deletes the key and its associated value. It returns the deleted value.

Example

```
delete $ salary {"employee1"};
```

The *delete* function can also be used to move an element from one hash to another as given below,

Example

```
$ new_sal {"emp2"} = delete $salary {"emp2"};
```

Since, the value deleted is returned, we can move it to another hash.

(b) Reverse

The *reverse* function is used to invert the hash. In general, a hash maps keys to values efficiently. However, if we want to find out whether the hash contains an entry for a particular value and map it onto its associated key then one of the solution is to iterate over the hash using foreach loop. A better solution is to invert the hash using the *reverse* function.

For example, a student hash with student number as the key and student name as value is inverted using *reverse* as follows,

```
%new_student = reverse %student;
$result = exists $new_student(name)?
$new_student(name) : "student name doesn't exist";
```

In the above example, the *reverse* function reverses the hash student into new_student with student name as the key and student number as value. The *exists* function checks whether an entry for given name is available in the hash or not. If it is available it returns the corresponding number otherwise returns a failure message.

1.15 Strings, Patterns and Regular Expressions

Q27. Define meta characters, character class and symbolic quantifiers.

Answer :

Meta Characters

Meta characters are those characters, which have specific meaning in patterns.

Some of the meta characters are tabulated below,

Meta character	Name	Meaning
●	Period	Matches only one character except a newline
|	Pipe	Matches one or the other alternatives
$	Dollar sign	True at end of string
^		True at beginning of string
\	Backslash	Matches de-meta next nonalphanumeric character, meta next alphanumeric character
( )	Parentheses	Matches a group of characters as a single unit
[ ]	Brackets	Matches a single character from a set of character class

Table: Meta Characters

Example

The pattern /Bone./matches "Boney", "Bonez" and "Bones".

In order to match a meta character itself, for instance, a period, then the period in the pattern is preceded by a backslash.

Example

/43.8/ matches 43.8.

Character Class

A character class, matches only a single character from a set. Characters classes are matched by placing the characters of interest in brackets.

Examples

- ❖ [pen] a characters class, which could match 'p', 'e' or 'n'.

  In order to invert a character class, we should prefix circumflex character (^) to a class of a characters.

- ❖ [^ PQR] matches all the characters, expect 'P', 'Q' and 'R'.

In order to repeat a characters or character class in a pattern, the numeric value should be written in the braces following the character.

Example

/Mob{2}ile/ matches mobile

Some character classes are used frequently, which can be specified by their names. They are shown in the table below,

Name	Matches
\d	A digit between 0-9
\D	Not a digit it may be a characters
\w	Alphanumeric [A-Z a-z_0-9)]
\W	Opposite to above not a alphanumeric character
\s	White space, horizontal tab newline, form feed, carriage return.
\S	Opposite to above [not a white space character]

Examples

/\d\t\d/ : It matches a digit followed by vertical tab and a digit.

/\w\w\w\w/ : It matches a word containing four alphanumeric characters.

/ \d\d\.\d/ : It matches two digits followed by a period and a digit.

Symbolic Quantifiers

The symbolic quantifiers are used, to repeat a pattern number of times. Perl supports three symbolic quantifiers. They are,

1. Asterisk (*): Repeats a pattern zero or more times.
2. Plus (+): Repeats a pattern one or more times.
3. Question mark (?): Repeats a pattern zero or none times.

Example

The pattern /p + q + z/, matches the string that starts with one or more p's followed by one or more q's, followed by z.

Symbolic quantifiers, can also be applied to character classes.

For example, the pattern /\d*\.\d/ matches the string that starts with zero or more digits followed period, followed by a digit.

A named pattern that matches the boundary position is \b (boundary), which is different from character classes.

It matches a position between two characters, a word character (\w) and a non-word character (\W).

Example : The pattern / \b do \b/ matches

"I do not want to go"

It doesn't match the string

"I don't want to go"

because "do" is followed by another character ('n').

Q28. What is the purpose of anchors and pattern modifiers in pattern matching?

Answer :

Anchors

Anchors are used to match a pattern at particular portion in a string. There are two anchors, which have specific meanings in this context of patterns.

(i) Circumflex (^) anchor

(ii) dollar-sign ($) anchor.

(i) Circumflex (^) Anchor

It is used to match a pattern, only at the starting of the string. Here, the string is preceded by (^).

Example

/^Sure/ matches "Sure publications"

/^Unix/ matches "Unix programming", "Unix operating system" etc.

(ii) Dollar-sign ($) Anchor

It is used to match only at the end of the string. The string here is succeeded by $.

Example

[good$]. which matches the string

"she is good"

[Answer $] which matches the string

"Right answer"

Anchors match positions before, between or after characters.

The anchors must be placed correctly at their position otherwise, they do not have special meanings.

Pattern Modifiers

To increase the flexibility of patterns, we can add modifiers.

Modifiers change the usage of patterns. They are denoted by letters and placed after the right delimiter of a pattern.

The modifier "*i*" is used, to match either uppercase or lowercase or any combination of uppercase and lowercase letters in the string.

Example

The pattern /data/i

matches 'DATA', 'DAta' 'data' etc.

The modifier provides a way to include-explanatory comments in the patterns, by allowing white space to appear in the pattern.

Example

```
[A–Z  a-z] +        #first name
[A–Z  a–z] +        #second name
/X
```

The above pattern is equivalent to

/[A – Z a – z _ 0 – 9] + [A – Z a – z _ 0 – 9] +/

Q29. Explain in detail about pattern matching with an example program.

Answer :

In Perl, operator 'm' is used as pattern matching operator. The slash delimited regular expression does not require '**m**' operator and it is a complete pattern-expression. Pattern matching expression, results in either true or false.

By default, the implicit variable, $_ is used for matching pattern against a string.

The value in the $_ is used for pattern matching in the following example,

```
if(/pen/) {
print
    "The word 'pen' appears in \$_ \n";
}
```

We can use the binding operator, = ~ for specifying the string, the string in $_ is not used in all the cases of pattern matching.

Let us see the example below,

```
if($str = ~ |^pen|)
{
print
    "The word 'pen' appears as the first word in the
    string \$str\n";
}
```

split function can take the first parameter as a single character or a pattern. As an example we take,

line_words = split /[.,]\s*/,$str,

The array @line_words, contain all the words from the string $str. Each word in string is separated, either by while space or comma or dot followed by one or more white spaces.

Example

A simple example of a program for a pattern matching Program read to file, in which the words are separated by punctuation marks[., ;] etc.

#If the word is encountered for the first time count is 1, otherwise count increments.

```
while(<>) {
    @words = split/[\.,;:!|?]\s*/;
foreach $word(@words) {
if(exists $count{$word}) {
    $count{word}++;
}
    else {
    $count{$word} = 1;
        }
    }
}
print "word\t count \n";
foreach $word(sort keys%count) {
print "$word t $count{$word} /n";
}
```

Q30. Write short notes on,

(a) Remembering matches

(b) Substitute operator

(c) Transliterate operator.

Answer :

(a) Remembering Matches

We can store the substring that matches some part of the pattern in an implicit variable, the part of the pattern must be placed in parentheses to save the result of this pattern. We can save the substring that matched the first parenthesis in $1, next in $2 and so on.

Example

```
"14 January 1987" = ~ /(\d+) (\w+) (\d+)/;
print "$3, $2, $1\n";
```

Output

1987 January 14

The three implicit variables $', $& and $', can be used after a match to get the substrings of the string that preceded, matched or followed the match respectively.

(b) Substitution Operator

Perl's substitute operator is used, to replace the part of the string that matched the pattern.

The substitute operator is represented as,

S/pattern/newstring/

Examples

❖ $str = "Sheela is a good girl"

$str = ~ S/good/bad;

Now $str becomes,

$str = "Sheela is a bad girl";

There are two modifiers, that can be used with the substitute operator. The modifier ***g*** of a substitute operator allows to replace all the matches in the string.

The modifier ***i*** allows to ignore the case of letters.

Example

(i) $str = "Rupees 10, 20, 300, 10";

$str = ~ S/10/100/g;

Now $str becomes,

$str = "Rupees 100, 20, 300, 100";

(ii) $str = "Madhu, Madhuri, Madhumathi";

$str = ~ S/Madhu/Sup/ig;

Now $str becomes,

$str = "Sup,Supri, SUPMATHI";

(c) Transliterate Operator

The translation of a character or character class to another character or character class can be done, using transliterate operator, which is denoted by, *tr*.

Syntax

tr/character/new_char;

Example

$str = ~ tr/./;/;

The above statement, replaces all the periods in the string with semicolons. This operation can also be done , using modifier ***g*** of substitution operator.

$str = ~ s/./;/g;

Translation is some what different from a substitution operation.

Let us take another example,

$str = ~tr/abc/ABC/;

The above statement translates **abc** in lowercase to uppercase.

We can also delete some character in the string, using null substitution

Example

$str = ~tr/;\!//;

The above statement delete. all; and ! from the string.

Q31. Explain pattern matching modifiers.

Answer : Model Paper-II, Q1(b)

Trailing qualifiers can change the usage of pattern match operator.

There are six pattern matching modifiers.

(a) m || i

This modifier is used to ignore the case when pattern matching. Here, 'i' denotes ignore case.

(b) m || g

This modifier is used to find all the occurences of the pattern. It can be used in two contexts. In a list context, all the substrings that match all the bracketed sections of the regular expression are returned.

In scalar context, the modifier iterates through a target string and returns either true of false.

If a match occurs it returns true otherwise false. This context is usually used in a while loop.

(c) m || m

In this, a string is assumed to contain more than one line, if it consists of new line characters. The start and end of the line is matched using anchors ∧ and $. Anchor '\A' indicates start of a string and anchor '\Z' indicates end of the string.

(d) m || s

This modifier is the reverse of the m || m. Here, even if a string contains new line characters it is considered as a single string. Any character including newline is matched with the dot.

(e) m || x

If white space characters are not escaped with a backslash or do not occur in a character class, they are ignored. This modifier is useful in adding a comment running to the next end-of-line by # (a metacharacter). With this modifier, the complex regular expressions can be easily made readable.

(f) m || o

This modifier ensures that the pattern is compiled only once. To perform a back-tracking match, the regular expression 'engine' needs to construct a non-deterministic finite automation. This is called compiling the regular expression. A regular expression is compiled when it is first encountered compiling a complex regular expression is time consuming.

When a pattern is encountered for the first time, which consists of variable substitutions, the modifier /o ensures that the values of variables occurring in a regular expression will not change and even if they are changed, they will have no effect.

The table below lists all the modifiers for pattern matching. Here, 'm' is the pattern match operator.

Modifier	Description
m‖i	Ignores case when pattern matching.
m‖g	Matches all the occurences of the pattern.
m‖m	Considers strings containing newline characters.
m‖s	Considers strings containing newline characters as a single line.
m‖x	Ignore whitespace characters
m‖o	Compiles the regular expression only once.

Table: Pattern Matching Modifiers

Q32. Explain with examples split and join functions.

Answer :

Split Function

The split function is used to split the given string. It can be used in two contexts.

In list context, the *split* function finds the delimiter in a string defined by the regular expression and returns the substring matched.

In scalar context, the split function returns the number of occurrences of the substrings.

Example

$Str = "Kavitha shekar uppleti" @line_words = split /\S+/, $Str;

In the above example, '$Str' string contains group of words separated by whitespace. The split function splits $Str into words are returns and returns a list of words in the array @line_words. If both the arguments that is the pattern and string are not present then whitespace is considered as delimiter and $_ as the target. (Default argument).

Join Function

This function returns a string constructed by concatenating two or more strings, with the separator provided as argument. It is the reverse of the split function.

Consider the previous example of split function. The words in the array (@line_words) separated by space are put together into the variable '$Str'.

$Str = join " ", @line_words;

Now, $Str contains "Kavitha Shekar Uppuleti".

1.16 Subroutines

Q33. Explain in detail about subroutines.

Answer : Model Paper-I, Q1(a)

Subroutine is a block of code having a name.

Declaration

To declare a subroutine, simply write *sub* before function name and enclose the code in bracess

```
Sub function_name
{
Statement 1
Statement 2
   ⋮     ⋮
Statement n
}
```

By observing the above subroutine, we notice that there no arguments are passed to subroutines. Infact, in Perl, a subroutine does not include arguments.

Return Expression

The return expression is used to return a specific value, this depends on the value sent to a subroutine (Scalar or list)

```
Sub function_name
    {
    Statement 1
    Statement 2
      ⋮    ⋮
    Statement n
    return value;
    }
```

Calling Subroutines

Subroutine can be called as,

```
&function_name;
    or
&function_name( );
```

'&' (ampersand) is used to identify the subroutine in the case when it is defined later. If in the script the subroutine is defined earlier then there is no need to mention '&' before the subroutine call.

Forward declarations of subroutines that are defined later in the script do not resume & at the time of calling that subroutine. A forward declaration of subroutine does not have a body. It is declared as follows,

```
Sub function_name;
```

Subroutine Arguments

Arguments can be passed in call to a subroutine as shown below,

```
&function_name(arg 1, arg 2, ....., arg n);
```

If the subroutine is declared earlier then '&' can be omitted.

```
function_name(arg 1, arg 2, ....., arg n);
function_name arg 1, arg 2, ..... , arg n
```

In conventional languages, the declaration of subroutines include the type & number of arguments. In Perl, this declaration carries variable number of arguments. The arguments of a subroutines stored in the anonymous array @–. Thus they can be accessed as $ –[0], $–[1] etc.

This passing of arguments has the effect of call by reference because the array @–stores implicit references to the actual scalar parameters. Thus, any changes to the elements of the array $–[0], $_[1], etc., effect the corresponding actual parameters.

The return value of a subroutine depends on the context in which the subroutine is called. Based on this context, the expression that determines the return value is evaluated. For example, if the subroutine is called in context as,

```
$a = myfun ($b, $c)
```

Then myfun returns a scalar value. If the subroutine is called in list context as,

```
@a = myfun($b, $c)
```

Then myfun returns a list of scalar values.

Q34. Write briefly about local and global variables.

Answer :

According to text_substitution model, global variables are the variables that are used in a subroutine. For example, if a variable $a exists then its value is taken by the same variable $a appearing in the body of the subroutine otherwise a new variable is created and initialized. This newly created variable is available for the rest of the script. To modify this behaviour of global variables, we use *local* (expression) or *my* (expression) declarations.

The variables declared with *my* declaration have lexical scope and these variables scope extends from the point of their declaration to the end of the block (innermost enclosing block). These variables exists only in the blocks in which they are declared. If a global variable has the same name as *my* declaration variable, then its functionality is hidden in that block i.e., we can declare, any number of variables in a *my* declaration.

Example

```
subthing
{
my $pen; my $pencil;
my ($a, $b, $c);
–
–
–
}
```

Using my declaration we can also initialize the variables as,

```
subthing1
{
    my($a, $b) = (20, 30)
    ....
}
```

my declaration can also be used to declare global variables. To achieve this effect, variables should be declared with *my* declaration at the start of a script.

OBJECTIVE TYPE

I. Fill in the Blanks

1. The process in which a collection of visual objects are used to build a graphical interface is called _____ scripting.
2. _____ are used to combine scalar data items into expressions.
3. _____ expression is used to select one value from the given two values at run time.
4. The meta characters which matches the start of a line is, _____.
5. A subroutine definition does not include _____ specifications.
6. A group of contiguous elements is called a _____.
7. The _____ function matches the given pattern with the given list.
8. The loop control commands used to force the next iteration of a loop and break out a loop are _____ and _____respectively.
9. The _____ and _____ commands are analogous to break and continue statements in C.
10. The pattern matching modifiers used to treat a string containing newline character as a single string is, _____.

II. Multiple Choice

1. Which of the following is not a characteristics of scripting languages? [ ]
   (a) Integrated compile and run (b) Enhanced functionality
   (c) More overheads and difficult to use (d) Compiled and run together
2. Which of the following is an application of modern scripting? [ ]
   (a) Web scripting (b) Visual scripting
   (c) Scripting compound document (d) All the above
3. String constants are enclosed in [ ]
   (a) Single quotes (b) Double quotes
   (c) Both (a) and (b) (d) None of the above
4. 'Any string' is equivalent to, [ ]
   (a) q / any string / (b) q (any string)
   (c) qq (any string) (d) Both (a) and (b)
5. Which one of the following is not a meta character? [ ]
   (a) / (b) (
   (c) * (d) ;
6. A meta character that matches any single character except new line is, [ ]
   (a) \ (b) .
   (c) ((d) *

7. Which of the following meta characters are called anchors? [ ]

   (a) ^ and | (b) ^ and $

   (c) $ and | (d) None of the above

8. Which of the following modifiers work with the substitution operator? [ ]

   (a) i (b) e

   (c) c (d) d

9. Which of the following is not a loop control command in Perl? [ ]

   (a) Last (b) Next

   (c) Undo (d) Redo

10. Which of the following manipulating lists return the first item of list? [ ]

    (a) Shift list (b) Unishift list

    (c) Push list (d) None of the above

KEY

I. Fill in the Blanks

1. Visual
2. Operators
3. Conditional
4. ^
5. Argument
6. Slice
7. Grep
8. Next, last
9. Last, next
10. M//s

II. Multiple Choice

1. (c)
2. (d)
3. (c)
4. (d)
5. (d)
6. (b)
7. (b)
8. (a)
9. (c)
10. (a)