Perl PROGRAMMING

A) **Perl Basics**

B) **Perl Variables**

C) **Perl Arguments**

D) **Perl Logic&Perl Looping**

E) **Perl Function**

F) **Perl and Files**

WWW.PRDEV.COM

مقدمه :

زبان برنامه نویسی Perl یک زبان مفسر می باشد که در سیستم عامل Unix بسیار محبوب می باشد.

علت محبوبیت و مقبولیت این زبان برنامه نویسی این است که این زبان دارای خصیوصیات و قدرتمند و قوی می باشد اما استفاده از آن بسیار ساده می باشد . این زبان (Perl) شباهت زیادی به زبانهای برنامه نویسی C و awk دارد و امکانات و انعطاف پذیری این زبانها را دارا می باشد و برای پردازشهای متنی ، تولید و گزارش گیری (Generating Reports) و Handling Common Gateway بسیار مفید و کاربردی می باشد.

Perl می تواند محیطهایی (محیطهایی که از سیستم عامل Unix استفاده نمی کنند) شاملMacintosh, Dos,OS/2 ,VMS,Windows NT مورد استفاده قرار گیرد و روی آنها به اجرا در آید . به عبارتی می توان گفت یکی دیگر از دلایل محبوبیت Perl این ویژگی است که زبانPerl می تواند با تغییرات اندکی روی بسیاری از Platform های روز دنیا به اجرا درآید . در این بخش ، مباحث اساسی در مورد زبان برنامه نویسی Perl را مورد بررسی قرار خواهیم داد :

WWW.PRDEV.COM

A) Perl Basics

برای ایجاد یک Perl Script ،از ویراستار (Editor) متنی برای وارد کردن دستورات Perl استفاده کنید و فایل را با پسوند pl. ذخیره کنید مانند Sample.pl . برای مشخص کردن اجرایی بودن فایل از chmod استفاده کنید .

در ادامه یک اسکریپت بیسار ساده را مورد بررسی قرار می دهیم :

```
#!user/bin/perl
print "I am here. \n";
```

در سطر اول اسکریپت، وجود # و ! و / برای به اجرا در آمدن تمامی اسکریپتهای Perl نیاز است .

#و ! و / به سیستم می گوید که برای اجرای دستورات اسکریپت، مفسر Perl را پیدا کند . در این اسکریپت شما معین کرده اید که پیغامی که در بین " " مشخص شده را برای شما چاپ کند .

از نکات مهم دیگر اسکریپت نویسی با Perl این است که اخر هر دستور باید سمی کالن قرار داد . همینطور \n این خاصیت را دارد که پس از چاپ شدن پیغام مورد نظر ، مفسر خط فرمان به خط بعد می رود ، درست برعکس دستورات echo در Bash Script .

اگر \n را در انتهای دستور چاپ پیغام قرار ندهید ، سایر پیغامها در ادامه همان خط چاپ می شوند .

اجرای Perl بر روی کامپیوتر :

پس از آنکه زبان Perl را روی کامپیوترتان نصب کردید ، برای استفاده و اجرای یک اسکریپتPerl ، ساده ترین راه این است که دستور زیر را تایپ کنید :

perl *filename*.pl

مثلاً درWin NTبرای اجرای یک اسکریپت به زبان Perl دستور زیر را در Command and Prompt تایپ کنید :

c:\perl5\bin\perl test.pl

و در سیستم عامل Unix دستور زیر را تایپ کنید :

```
#!/usr/local/bin/perl
```

WWW.PRDEV.COM

: Statement Blocks

بلوکهای برنامه های به زبان Perl بوسیله یک brace باز شروع می شوند و با یک brace بسته خاتمه می یابند .

```
{

   statement_1;
   statemrnt_2;
   statemrnt_3;

.....

   statement_n;
}
```

فرمتهای مختلف نمایش رشته :

\n	**خط بعدی**
\t	**پرش به جلو به اندازه یک** Tab
\b	**به اندازه یک کاراکتر به عقب می رود**
\\	**نمایش** \
\"	**برای نمایش " در خروجی**
\l	**نمایش کاراکتر بعدی بصورت** lower case
\L	**کلیه کاراکترهای بین** \L **تا** \E **بصورت** lower case **نمایش داده می شوند**
\u	**نمایش کاراکتر بعدی بصورت** upper case
\U	**کلیه کاراکترهای بین** \L **تا** \E **بصورت** upperr case **نمایش داده می شوند**
\E	Terminate \L or \E

: printf

این تابع برگرفته شده از زبان برنامه نویسی c می باشد که دقیقاً همانند printf زبان c عمل می کند :

در ادامه بحث لیست کوتاهی از انواع شاخصهای زمینه printf را خواهیم گفت :

نوع زمینه	معنا
s	نمایش بصورت کاراکتری
s	نمایش بصورت رشته ای
d	قسمت صحیح اعشار ، اعداد گرد شده اند
f	عدد با نقطه شناور

WWW.PRDEV.COM

چند مثال :

```
$amt=7.12;
printf("%6.2f");
output : 7.12

printf("%c",65);
output : A

printf("06.2f",$amt);
output : 007.12
```

WWW.PRDEV.COM

B) Perl Variables

متغیرها در Perl به چهار دسته تقسیم می شوند :

۱- متغیرهای عددی
۲- متغیرهای کاراکتری
۳- آرایه های اسکالر
۴- آرایه های شرکت پذیر

در Perl شما ملزم نیستید Type برای متغیر خود مشخص کنید ، بلکه در زمان مقداردهی به متغیرها در طول برنامه معین می شود که آیا متغیری که تعریف کرده بودید از نوع عددی بوده یا کاراکتری (یا دنباله ای از کاراکترها یا رشته ای) . همچنین از قابلیتهای دیگر متغیرها در Perl این است که Type متغیرها قابل تبدیل می باشند یعنی مثلاً یک متغیر عددی می تواند به کاراکتری تبدیل شود و بالعکس متغیر کاراکتری به عددی .برای تعیین کردن متغیرها و دسترسی به مقدار آنها ، از کاراکتر پیشوندی $ استفاده می شود . Space بعد و قبل = اختیاری است و مشکلی در اسکریپت بوجود نمی آورد . اما باید توجه داشت که در مقداردهی به متغیرهایی که میخواهیم Type رشته ای داشته باشند ، باید رشته را بین دو " قرار دهیم . با ذکر ۲ مثال مفاهیم اولیه در مورد قواعد متغیرها مورد بررسی قرار می دهیم :

```
$num=5;
$num= 5;
$num = 5;
```

هر سه شکل درست است . توجه شود که در آخر هر دستور در Perl سمی کالن باید قرار داده شود.

هر سه دستور یک عمل واحد را انجام می دهند ک: مقدار متغیر num را ۵ قرار می دهند .

```
$stuff = "I am here";
```

دستور بالا هم عمل assign کردن را انجام می دهد ، یعنی محتوای متغیر stuff را با مقدار رشته ای I am here ست می کند .

```
$user = $ENV{USER};
```

دستور بالا مقدار متغیر USER environment را در user قرار می دهد .

WWW.PRDEV.COM

کلاسهای کاراکترها :

***) کاراکترهایی که شامل [0123456789] می باشند نمایانگر اعداد می باشند .**

***) کاراکترهایی که شامل کاراکترهای [0-9a-f] می باشند نمایانگر اعداد هگزادسیمال می باشند .**

***) کاراکترهایی که شامل کاراکترهای [0-7] می باشند نمایانگر اعداد اکتال می باشند .**

```
$octal_number = 0377; # مثل ۲۵۵ دسیمال
$hex_number = 0xff; # مثل ۲۵۵ دسیمال
```

توجه : داده های هگزا دسیمال و اکتال با ۰ شروع می شوند مانند :

```
$octal_number = 0377;
$hex_number = 0xff;
```

در Perl چند مدل می توان از Quotation ها استفاده کرد که شامل Single Quotation ، Double Quotation ، Backticks و Backslash می باشد .

Single Quotation :

```
$user = $ENV{USER};
print 'Good Morning $user';
----------
Out put : Good Morning $user
```

همانطور که در این مثال دیده می شود ، $user عیناً در خروجی چاپ شده است ، پس اینگونه بیان می کنیم که : یکی از موارد استفاده Single Quotation زمانی است که می خواهیم عبارت داخل Quotation را عیناً در خروجی چاپ کنیم .اگر عبارتی که می خواهیم عیناً در خروجی چاپ کنیم در Perl دارای معنی و مفهوم خاصی باشد مانند کاراکتر $ ، باید از Single Quotation استفاده شود.

Double Quotation :

```
$user = $ENV{USER};
print "Good Morning $user";
----------
```

WWW.PRDEV.COM

Out put : Good Morning Mahdi

همانطور که می بینید ، **Double Quotation** زمانی استفاده می شود که می خواهید یا عبارت عیناً چاپ شود یا مقدار یک متغیری را چاپ کند که دارای مقدار خاصی می باشد مثل :

```
$user = $num = 5;

print "Number is $num";
```

Out put : Number is 5

Bachticks :

```
$currdir = 'pwd';

print "Currect directory is $pwd";
```

Out put : Currect directory is /home/Mahdi

Backticks را زمانی استفاده کنید که رشته مورد نظر شامل یک **Unix Command** (دستور **Unix**) می باشد .

Backslash :

```
$costmsg = "Price is \$5.00";

print "$costmsg";
```

Out put : Price is $5.00

این نماد زمانی مورد استفاده قرار می گیرد که بخواهیم یک کاراکتر تک که معنی خاصی دارد مثل مثلاً کاراکتر **$** را نادیده بگیرد ، زیرا در غیر این صورت ممکن است کاراکتر خاص مورد نظر که می خواهیم عیناً در خروجی چاپ شود ، فرمت خروجی ما را تغییر دهد .

توابع رشته ای :

substr(STRING, OFFSET, LENGTH) :

این تابع قسمتی از رشته (**STRING**) را که در پارامتردوم (**OFFSET**) به آن داده شده است و طول رشته یا **LENGTH** را به ما می دهد . اگر طول در پارامترهای ورودی تعیین نشود آنگاه تمامی رشته از **FFSET**

WWW.PRDEV.COM

تا انتها را به ما می دهد .

توجه کنید که اگر مقدار منفی به OFFSET داده شود آنگاه مقدار رشته از انتها به ابتدا را به ما می دهد :

```
$firstVar = "0123BBB789";
substr($firstVar, 4, 3) = "AAA";
print("firstVar  = $firstVar\n");
```

This program prints:

```
firstVar = 0123AAA789
```

join(STRING, ARRAY) :

خروجی این تابع رشته ای است که تمامی آیتمهای آرایه با رشته داده شده ادغام شده است . برای درک بهتر مطلب مثالی آورده شده است :

```
join(">>", ("AA", "BB", "cc"))
 returns "AA>>BB>>cc".
```

lcfirst(STRING) :

این تابع رشته ای را برمیگرداند که اولین کاراکتر آن (اگر حرف بزرگ انگلیسی باشد) به حرف کوچک انگلیسی تبدیل شده است یعنی اولین حرف رشته را **Lower Case** می کند :

```
lcfirst("ABCD")
 returns "aBCD"
```

Ucfirst(STRING) :

مانند **lcfirst()** می باشد با این تفاوت که حرف اول رشته را **Upper Case** می کند :

```
Ucfirst("abcd")
 returns "Abcd"
```

lc(STRING) :

تمامی کاراکترهای رشته ورودی را **Lower Case** می کند :

```
lc("ABCD")
```

WWW.PRDEV.COM

```
returns "abcd".
```

uc(STRING) :

مانند **lc()** می باشد با این تفاوت که تمامی کاراکترهای رشته را **Upper Case** می کند :

```
uc("abcd")
returns "ABCD".
```

chr(NUMBER) :

این تابع معادل اسکی عدد ورودی (**NUMBER**) را به ما می دهد :

```
chr(65)
returns the letter A.
```

length(STRING):

این تابع طول رشته ورودی را به ما می دهد :

```
Length("abs");
Return 3.
```

chop(STRING) OR chop(ARRAY) :

این تابع کاراکتر آخری رشته ورودی یا آرایه ورود ی را حذف می کند و کاراکتر حذف شده را بعنوان خروجی تابع برگشت می دهد .

```
$str = "sssa";
Chop($str);
Return "a"      $str ="sss" .
```

index(STRING, SUBSTRING, POSITION) :

این تابع از سمت چپ **STRING** آغاز می گردد و به دنبال یک **SUBSTRING** می گردد . تابع **index** وضعیت را به جایی باز می گرداند که **SUBSTRING** یافت می شود . **STRING** می تواند یک رشته حرفی یا اسکالر و یا هر عبارتی که یک مقدار رشته ای را باز می گرداند باشد .

توجه شود که چنانچه **SUBSTRING** در **STRING** یافت نشود مقدار بازگشتی **1-** خواهد بود :

WWW.PRDEV.COM

Index(“I am a Student”,”am”)

Return 2

آرایه های اسکالر :

آرایه های اسکالر به صورت زیر تعریف می شوند :

```
@array1 = (4,5,6);
@array2 = (1,2,3, @array1, 7,8);
```

توابع کار با آرایه های اسکالر :

Push() و pop() :

تابع push() یک عنصر به سمت راست یک آرایه اضافه می کند و تابع pop() یک عنصر از سمت راست یک آرایه حذف می کند :

```
push(@array1,7);
# یک عنصر به سمت راست آرایه اضافه می کند
Pop(@array1)
# یک عنصر از سمت راست آرایه حذف کرد
```

همانطور که مشاهده می کنید ، push() دو پارامتر دارد در صورتیکه pop() یک پارامتر دارد .

shift() and unshift() :

همانند push() و pop() می باشد با این تفاوت که یک عنصر به سمت / از سمت راست آرایه اضافه / حذف می شود .

reverse() :

عناصر آرایه را برعکس می کند به این صورت که :

```
@a = (1,2,3);
@b = reverse(@a);
```

آرایه @b شامل عناصر مقابل خواهد بود : (3,2,1)

Sort() :

WWW.PRDEV.COM

```
@x = sort("small","medium","large");
# gets @x = ("large","medium","small");
@y = sort(1,,32,16,4,2);
# gets @x = (1,16,2,32,4);
```

آرایه های شرکت پذیر یا انجمنی (Associative Arrays) :

آرایه های شرکت پذیر یکی از پرکاربردترین نوع از متغیرهای مورد استفاده در زبان **Perl** می باشد .

آرایه های شرکت پذیر بطور کلی جداول اطلاعاتی رل ذخیره می کند .

Hashes (هش ها) :

سومین نوع داده ای موجود در **Perl** می باشد .

هش ها ، نوع دیگری از داده های گروهی هستند . هش ها هم مانند آرایه ها ، یک عدد اسکالر را در خود وارد نموده اند . تفاوت آرایه ها و هش ها این است که هش ها به داده اسکالر خود از طریق نام دسترسی پیدا می کنند ، ولی آرایه ها از زیر نویس عددی استفاده می کنند . عناصر هش دو قسمت دارند :

۱ –کلید

۲ –مقدار .

کلید ، هر عنصر هش و مقداری که با آن کلید به داده مربوط می شود را مشخص می نمایند . این رابطه جفت «کلید –مقدار» نامیده می شود .

طبیعتاً بسیاری از برنامه های کاربردی خود را با این نوع ساختار وقف داده اند . بعنوان مثال ، چنانچه خواهید اطلاعات مربوط به رانندگانی که در یک شهر گواهینامه رانندگی دارند را ذخیره نمایید ، باید شماره گواهینامه را بعنوان کلیدهای منطقی بکار ببرید تا اطلاعات مربوط به گواهینامه را ذخیره نمایید . این شماره ها (بازای هر راننده) منحصز به فرد می باشد . داده هایی که به هر شماره مربوط می شوند ، اطلاعات مربوط به آن راننده می باشد . (مثل گواهینامه ، سن ، آدرس و ...) . هر گواهینامه رانندگی ، عنصری را در هش نشان می دهد که همراه آن عنصر ، شماره و اطلاعات مربوطه بعناون جفت کلید –مقدار قرار دارند .

WWW.PRDEV.COM

name	code
dave	1234
peter	3456
andrew	6789

در **Perl** ، هش می تواند هر عنصری را که بخواهید و یا حداقل تا اندازه ای که حافظه سیستم به شما اجازه دهد ،در خود بگنجاند . هنگامی که عنصری به هش اضافه می شود ، اندازه هش تغییر می کند . دسترسی به داده های خاص در هش بسیار سریع صورت می گیرد . و هنگامی که هش بزرگتر می شود ، این سرعت یعنی سرعت دسترسی به داده ها کاهش نمی یابد .

در **Perl** و زبانهای دیگر ، از قدیم الایام هش ها را «آرایه های شرکت پذیر یا انجمنی» می نامیدند . این اصطلاح یک روش طولانی برای نمایش آن کلیدهایی است که به یک مقدار مربوط می شوند . اکنون آرایه های شرکت پذیر یا انجمنی را فقط هش می نامند ، زیرا برنامه نویسان **Perl** به طولانی بودن آن علاقه نداشتند .

متغیرهای هش در **Perl** با علامت درصد **(%)** مشخص شده اند . نام آنها را نمی توان در آرایه ها و داده های اسکالر بکار برد . بعنوان مثال ، می توانید یک هش به نام **%A** و یک آرایه به نام **$A** را داشته باشید . هر یک از این نامها به متغیری مربوط می شوند که هیچ ربطی به هم ندارند .

ایجاد یک هش :

```
%lookup = ("dave", 1234,
           "peter", 3456,
           "andrew", 6789);
```

عملگرهای هش :

()Keys : این عملگر کلیدهای یک هش را برای ما لیست می کند :

```
@names = keys(%lookup);
```

()values : این عملگر مقدار یک هش را به ما می دهد :

```
@codes = values(%lookup);
```

()delete : این عملگر زوج کلید-مقدار یک هش را حذف می کند :

```
delete %lookup("adam");
```

ذخیره رکوردی از داده ها با استفاده از هش :

یکی از کاربردهای بسیار عالی هش ها در زبان برنامه نویسی **Perl** ذخیره رکوردی از داده هاست .

ذخیره داده ها و همچنین بازیابی آن نیز بسیار آسان خواهد بود . به مثال زیر توجه کنید :

```
%record = ( "Name"    => "Jane Hathaway",
            "Address" => "123 Anylane Rd.",
            "Town"    => "AnyTown",
            "State"   => "AnyState",
            "Zip"     => "12345-1234"
);
```

در مثال بالا ما رکوردی از داده های یک مشتری را ذخیره کردیم .

این رکورد ۵ فیلد اطلاعاتی دارد .

در این مثال شما می توانید از طریق این فیلدهای اطلاعاتی به مقادیر آن دسترسی داشته باشید مثلاً :

```
$record{"State"}.
```

حال با یک مثال کاربردی مطالب بیشتر روشن خواهد شد :

```
%recordOne = ( "Name"    => "Jane Hathaway",
               "Address" => "123 Anylane Rd.",
               "Town"    => "AnyTown",
               "State"   => "AnyState",
               "Zip"     => "12345-1234"
);

%recordTwo = ( "Name"    => "Kevin Hughes",
               "Address" => "123 Allways Dr.",
               "Town"    => "AnyTown",
               "State"   => "AnyState",
               "Zip"     => "12345-1234"
);
```

WWW.PRDEV.COM

```
@database = ( \%recordOne, \%recordTwo );
```

شما می توانید آدرس رکورد اول را چاپ کنید :

```
print( %{$database[0]}->{t;{"Address"} . "\n");
```

خروجی مطلوب شما خواهد بود :

```
123 Anylane Rd.
```

استفاده از (..) در تعیین محدوده :

در Perl از .. برای تعیین محدوده آرایه ها استفاده می شود که از این روش برای تندنویسی استفاده می شود

مثلاً: @array = (1..10) شامل ۱و۲و۳و۴و۵و۶و۷و۸و۹و۱۰ می باشد .

```
@array = ("A".."J")  including of A,B,C,D,E,F,G,H,I,J
@array = ("01".."10") including of 01,02,03,04,05,06,07,08,09,10
@array = ("aa" .. "af")including of aa, ab, ac, ad, ae, and af
```

%ENV Variable :

Environment Variable متغیری است که توسط سیستم عامل بیتهای اطلاعاتی را که در برای راه اندازی و اجرای کامپیوتر نیازند ذخیره می کند .

علت نامگذاری این متغیرها با Environment Variable این است که گاهی از آنها در کامپیوتر استفاده می شود .

برنامه کاربردی زیر روش استفاده از %ENV و محتوای آن را پس از اجرا نشان می دهد :

```
foreach $key (keys(%ENV)) {
    printf("%-10.10s: $ENV{$key}\n", $key);
}
```

با اجرای این کد در Win95 نتیجه بصورت زیر خواهد بود :

```
WINBOOTDIR: C:\WINDOWS
TMP       : C:\WINDOWS\TEMP
```

WWW.PRDEV.COM

```
PROMPT    : $p$g

CLASSPATH : .\;e:\jdk\classes;

TEMP      : C:\WINDOWS\TEMP

COMSPEC   : C:\WINDOWS\COMMAND.COM

CMDLINE   : perl -w 12lst01.pl

BLASTER   : A220 I10 D3 H7 P330 T6

WINDIR    : C:\WINDOWS
PATH      : C:\WINDOWS;C:\WINDOWS\COMMAND;C:\PERL5\BIN;
TZ        : GMT-05:00
```

سایر متغیرهای ویژه در Perl :

سایر متغیرهای ویژه ای که در زبان برنامه نویسی Perl کاربرد دارند را (تعدادی از آنها را) مشاهده خواهید کرد :

$_ : پارامتر پیش فرض برای بسیاری از توابع

$# : فرمت پیش فرض برای اعداد چاپ شده

$0 : نام فایلی را در خود دارد که فایل مورد نظر شامل Script اجرا شده توسط زبان Perl می باشد .

$@ : حاوی پیغام syntax error می باشد .

$^X : شامل مسیر کامل مفسر زبان Perl زمانیکه یک script صحیح را اجرا می کند .

WWW.PRDEV.COM

C) Perl Arguments

Arguments and other Special Variables :

آرگومانها مقادیری هستند که ما در زبان **Perl** به اسکریپتها پاس می دهیم .هر مقداری در هر خط از دستور العمل بعد از نام اسکریپت به متغیر خاصی **Assign** خواهد شد . شماره آرگومانهای پاس داده شده در اسکریپت در متغیر خاصی به نام **$#ARGV** ذخیره می شود . و آرگومان رشته ای اصلی در متغیر در **@ARGV** قرار دارد . همچنین نام برنامه اخیری که اجرا شده است در متغیر **$0** قرار دارد .

حال یک فایل اجرایی به نام **testvars.pl** بر روی سیستم می سازیم (طریقه ساخت آن گفته شده) و دستورات زیر بهمراه خروجی اجرای آنها را مورد بررسی قرار می دهیم تا مطالب گفته شده را بهتر درک کنید

```
#!/usr/bin/perl
print "My name is $0 \n";
print "First arg is: $ARGV[0] \n";
print "Second arg is: $ARGV[1] \n";
print "Third arg is: $ARGV[2] \n";
$num = $#ARGV + 1; print "How many args? $num \n";
print "The full argument string was: @ARGV \n";
```

Out put :

```
$ ./testvars dogs can whistle

My name is testvars.pl

First arg is: dogs

Second arg is: can

Third arg is: whistle

How many args? 3

The full argument string was: dogs can whistle
```

نکته قابل توجه این است که همانطور که در محاسبه **How many args** می بینید ، برای محاسبه آن **$#ARGV + 1** استفاده شد ، یعنی **$#ARGV** را با یک جمع کردیم ، علت چیست ؟

اگر با زبان برنامه نویسی **c** کار کرده باشید می دانید که شماره آرایه ها و آرگومانها از صفر شروع می شود

WWW.PRDEV.COM

در Perl هم از این قاعده استفاده می شود و برای همین $#ARGV با یک جمع شد تا تعداد واقعی $#ARGV بدست آید .

نکته قابل توجه دیگر در مورد @ARGV می باشد که با کاراکتر $ شروع نشده است بلکه با @ شروع شده است (همانطور که گفته شد متغیرها در اسکریپتها با کاراکتر $ مشخص می شوند) . علت این است که @ARGV آرایه ای از متغیر هاست . آرایه ها لیستی از مقادیر هستند که هر یک از مقادیر این لیست با استفاده از کاراکتر $ و ایندکس آن قابل دسترسی می باشد :

@ARGV is an array that is including $ARGV[1] , $ARGV[2] , ... , $ARGV[n]

WWW.PRDEV.COM

D) Perl Logic&Perl Looping

Conditional Operations

فرمت و شکل کلی **if – then – else** در ادامه بحث مورد بررسی قرار می گیرد :

```
if ( condition is true ) {
execute these commands
}
else {
execute those commands
}
```

در شکل کلی که در بالا آمده است وجود شرط**else** اختیاری می باشد . عبارت **if – then – else** بصورت زیر نیز می تواند باشد :

```
if ( condition is true ) { execute these commands }
elsif {execute those commands }
elsif {execute those commands }
else { execute those commands }
```

Loop Operations

در ادامه بحث ساختار حلقه های **While - For** - مورد بررسی قرار می گیرد :

While loop :

ساده ترین نوع حلقه ، **حلقه While** می باشد که تا زمانیکه شرط حلقه درست باشد ، اجرا می شود :

```
for ( initialise_expr; test_expr; increment_expr )
  {
     statement(s);
  }
```

مثال :

```
$a = 0;

While ($a < 10 ) {

  Print "Still continue … $a\n";

  $a++;
```

WWW.PRDEV.COM

}

For Loop:

در این شکل از حلقه به تعداد عناصر یک لیست تکرار می شود :

```
for ( initialise_expr; test_expr; increment_expr )
  {
     statement(s);

 }
```

مثال :

```
for ( $i = 1; $i <= 10; ++$i )
  { # count to 10
     print "$i\n";
  }
```

Until Loop :

نوع عملکرد این نوع از حلقه های تکرار مانند While می باشد با این تفاوت که While تا زمانی دستورات آن اجرا می شود که شرط حلقهTrue باشد اما Until تا زمانی انجام می شود که شرط حلقه False باشد :

```
i = 1;
until (  $i > 10 )
  { # count to 10
     print "$i\n";
     ++$i;
  }
```

WWW.PRDEV.COM

عملگرهای منطقی در Perl :

And :

نماد این عملگر در زبان برنامه نویس Perl بصورت زیر می باشد :

```
op1 && op2
```

Or :

نماد این عملگر بصورت زیر می باشد :

```
op1 || op2
```

: Not

نماد این عملگر بصورت زیر می باشد :

```
!op1
```

مثالهای عملگرهای منطقی :

```
if ($firstVar == 10 && $secondVar == 9) {

    print("Error!");

};
$firstVar = 10;

$secondVar = !$firstVar;


if ($secondVar == 0) {

   print("zero\n");

};
```

WWW.PRDEV.COM

عملگرهای بیتی :

این نوع از عملگرها شباهت زیادی به عملگرهای منطقی دارند . بجز اینکه این نوع عملگرها در مقیاس کوچکتری اعمال می شوند –در نمایش باینری داده ها .

`op1 & op2` -- **The AND operator :**

این عملگر دو بیت را با هم مقایسه می کند و در صورتی که هر دو بیت یک باشند نتیجه ۱ می باشد و در غیر این صورت نتیجه برگشت داده شده ۰ خواهد بود .

`op1 | op2` -- **The OR operator :**

این عملگر دو بیت را با هم مقایسه می کند و در صورتی که هر دو بیت متمم یکدیگر باشند نتیجه ۱ می باشد و در غیر این صورت نتیجه برگشت داده شده ۰ خواهد بود .

`op1^ op2` -- **The EXCLUSIVE-OR :**

یا عملگر or انحصاری که یا xor که در صورتیکه بیتهای هر دو عملگر یا هیچکدام از بیتهایشان ۱ نباشد و یا همه بیتهایشان ۱ باشد و در غیر این صورت نتیجه برگشت داده شده ۰ خواهد بود .

~op1 -- The COMPLEMENT :

این عملگر تمامی بیتهای متغیر را عکس می کند . ۰ ها را تبدیل به ۱ و ۱ ها را تبدیل به ۰ خواهد کرد .

op1 >> op2 -- The SHIFT RIGHT :

این عملگر op1 را به تعداد op2 به سمت راست شیفت می دهد .

op1 << op2 -- The SHIFT LEFT :

این عملگر op1 را به تعداد op2 به سمت چپ شیفت می دهد .

برای درک بهتر مطالب گفته شده به دو مثال ذکر شده توجه کنید :

```
$firstVar = 128;
$secondVar = $firstVar << 3;
print $secondVar;
```

خروجی برنامه خواهد بود :

```
1024
```

```
$firstVar = 128;
$secondVar = $firstVar >> 2;
print("$secondVar\n");
```

خروجی برنامه خواهد بود :

```
32
```

WWW.PRDEV.COM

عملگرهای مقایسه ای در Perl :

مقایسه ای	عددی	رشته ای
Equal	==	eq
Not Equal	!=	ne
Comparison	<=>	cmp
رابطه ای	عددی	رشته ای
Less than	<	lt
Greater than	>	gt
Less than or equal	<=	le
Greater than or equal	>=	ge

Cmp = Compare(مقایسه)

Lt = Little(کوچکتر)

Gt = greater(بزرگتر)

Ls = less

چند مثال از عملگرهای مقایسه ای و رابطه ای :

```
if ( ($var ne "0") && ($var == 0))
  {
  }
```

$var یک مقدار عددی می باشد .

```
$lowVar =  8;

$midVar = 10;

$hiVar  = 12;

print($lowVar <=> $midVar, "\n");

print($midVar <=> $midVar, "\n");

print($hiVar  <=> $midVar, "\n");
```

WWW.PRDEV.COM

خروجی برنامه به شکل زیر خواهد بود :

```
-1
0
1
```

خروجی 1- نشان می دهد که $lowVar (8) کوچکتر از $midVar (10) می باشد .

```
$lowVar = "AAA";

$midVar = "BBB";

$hiVar  = "ccC";

print($lowVar cmp $midVar, "\n");

print($midVar cmp $midVar, "\n");

print($hiVar  cmp $midVar, "\n");
```

خروجی برنامه بصورت زیر خواهد بود :

```
-1
0
1
```

عملگرهای ریاضی :

+ : $x = 3 + 1; $y = 6 - $x;(جمع)

- : $x = 3 + 1; $y = 6 - $x; (تفریق)

*** :** $z = $x * $y;(ضرب)

/ : $z = $x * $y;(تقسیم)

** : $w = 2**3; # 2 to the power of 3 = 8(به توان رساندن)

% : $z = $x % $y;(باقیمانده)

++ : $a = 3;$b = ++$a;**(inc a)**

-- : $a = 3;$b = --$a;**(dec a)**

WWW.PRDEV.COM

عملگرهای رشته ای :

زبان برنامه نویسی Perl دو عملگر مهم رشته ای دارد :

۱ – عملگر اتصال (.) : عملگر «.» برای متصل کردن دو رشته بکار می رود :

```
$first_name = "David";
$last_name = "Marshall";
$full_name = $first_name . " " . $last_name;
print ($full name);
```

خروجی خواهد بود :

David Marshall

۲ - عملگر تکرار رشته (X) : این عملگر «X» برای تکرار رشته بکار می رود :

```
$first_name = "David";
$david_cubed = $first_name x 3;
print($david_cubed);
```

خروجی خواهد بود :

```
DavidDavidDavid
```

این برنامه David را ۳ بار در خروجی تکرار کرده است .

WWW.PRDEV.COM

E) Perl Functions

توابع بلاکی از کدها(کد) می باشند که شما می توانید در طول برنامه اصلیتان بوسیله نامی که برای تابع مورد نظرتان انتخاب کرده اید آن را صدا زده و از آن استفاده نمایید .

توابع این امکان را فراهم می سازند که بتوانید برنامه اصلی رل به قطعاتی از کد جدا کنید تا راحتتر بتوانید آنرا تست کنید .

تعریف تابع در Perl :

```
sub functionName {

                  }
```

همچنین توابع می توانند مقادیری را بعنوان ورودی تابع داشته باشند .

پارامترهای ورودی تابع در یک آرایه به نام _@ذخیره می شوند .

```
$areaOfFirstCircle = areaOfCircle(5);
print("$areaOfFirstCircle\n");
sub areaOfCircle {
    $radius = $_[0];
    return(3.1415 * ($radius ** 2));
}
```

This program prints:

```
78.7375
```

توابع با پارامتر ورودی Array :

همانطور که گفته شد پارامترهای ورودی توابع در Perl در آرایه _@ ذخیره می شوند . یکی از مزایای آن این است که برنامه نویس می تواند دریابد که چه تعداد پارامتر به تابع ای که تعریف کرده پاس داده است .

برای درک بهتر مطلب گفته شده به مثال زیر توجه کنید :

```
firstSub(1, 2, 3, 4, 5, 6);
firstSub(1..3);
```

WWW.PRDEV.COM

```
firstSub("A".."Z");


sub firstSub {
    $numParameters = @_ ;
    print("The number of parameters is $numParameters\n");
}
```

خروجی این برنامه خواهد بود :

```
The number of parameters is 6
The number of parameters is 3
The number of parameters is 26
```

همانطور که مشاهده می کنید یکی از مزایای زبان برنامه نویسی **Perl** این است که هر تعداد پارامتر میتوانید به تابع ای که تعریف کرده اید پاس دهید .

اما زبان **Perl** این امکان را هم به شما می دهد تا بتوانید مقادیر عددی را به تابع ای که تعریف کرده اید پاس دهید . با ذکر مثال زیر روش این کار را بیان می کنیم :

```
areaOfRectangle(2, 3);
areaOfRectangle(5, 6);
sub areaOfRectangle {
    ($height, $width) = @_ ;
    $area = $height * $width;
    print("The height is $height. The width is $width.
        The area is $area.\n\n");
}
```

خروجی این برنامه خواهد بود :

```
The height is 2. The width is 3.
```

WWW.PRDEV.COM

```
        The area is 6.
The height is 5. The width is 6.
        The area is 30.
```

زبان برنامه نویسی **Perl** این اجازه را به شما می دهد که هر مقدار عددی را بعنوان پارامتر ورودی به تابعی که تعریف کرده اید پاس دهید . در تابع می توانید تصمیم بگیرید از کدام پارامتر استفاده کنید .

References :

ارجاعت مقادیر عددی هستند که ناحیه ای از فضای حافظه اشاره می کنند که می توانند انواع مقادیر داده ای را در خود جای دهند . هر چیزی در برنامه ای که شما به زبان **Perl** می نویسید در قسمتی از حافظه کامپیوترتان ذخیره می شود . ارجاعات یا **Refrence** ها زمانی استفاده می شوند که شما بخواهید محتوای قسمتی از حافظه تان را بازیابی کنید .

انواع Refrence ها :

در زبان **Perl** ما ۶ گونه مختلف **Refrence** داریم که در جدول زیر مشاهده می کنید :

Reference Assignment	*How to Dereference*
`$refScalar = \$scalar;`	${$refScalar} یک مقدار عددی است
`$refArray = \@array;`	@{$refArray} یک مقدار آرایه ای است
`$refHash = \%hash;`	%{$refHash} یک مقدارهش است
`$refglob = \*file;`	Glob references are beyond the scope of this course
`$refFunction = \&function;`	&{$refFunction} is a function location.
`$refRef = \$refScalar;`	${${$refScalar} یک مقدار عددی است

WWW.PRDEV.COM

پاس دادن پارامترها به توابع :

زمانیکه ما تابعی را در طول برنامه تعریف کردیم (همانطور که گفته شد) بیش از یک آرایه را نمیتوانیم به تابع پاس دهیم . دلیل آن است که توابع تنها یک آرایه _@ (آرایه _@) را بعنوان پارامتر ورودی میشناسد . برای غلبه بر این ضعف Perl از Refrence ها استفاده می شود :

```
firstSub( (1..5), ("A".."E"));
sub firstSub {
    my(@firstArray, @secondArray) = @_ ;
    print("The first array is  @firstArray.\n");
    print("The second array is @secondArray.\n");
}
```

This program displays:

```
The first array is  1 2 3 4 5 A B C D E.
The second array is
```

همانطور که می بینید تابع تعریف شده دو آرایه پاس داده شده را به شکل یک پارامتر ورودی می بیند.

با پاس دادن Refrence ها به `@arrayOne and @arrayTwo` ما دو آرایه را در قالب یک آرایه به تابع شناساندیم .

حال این کار را (پاس دادن ۲ آرایه به تابع) را با استفاده از `pass_ref.pl` انجام می دهیم .

با این روش نیز می توان ۲ آرایه را به تابع پاس داد . این روش را با یک مثال توضیح می دهیم :

```
firstSub( (1..5), ("A".."E"));        # One

sub firstSub {
    my($ref_firstArray, $ref_secondArray) = @_    # Two

    print("The first array is  @{$ref_firstArray}.\n");  # Three
    print("The second array is @{$ref_secondArray}.\n"); # Three
}
```

WWW.PRDEV.COM

خروجی برنامه به صورت زیر خواهد بود :

```
The first array is  1 2 3 4 5.
The second array is A B C D E.
```

تعریف متغیرهای محلی در توابع :

با استفاده از اپراتور Local می توانید متغیری را محلی که فقط در محدوده تابع تعریف شده شناخته شده است تعریف کنید :

```
Sub myfunc {
        Local($foo) = 56;
}
```

تفاوتی متغیرهای محلی با سراسری این است که این متغیر در محدوده بلوک محدود شده اش استفاده کرد و پس از خروج از محدوده تعریفش از بین می رود .

Private Function (توابع خصوصی) :

بعضی مواقع ، در زبان برنامه نویسی Perl ، برنامه نویس می خواهد از توابع خصوصی استفاده کند .

به مثال زیر توجه کنید :

```
$temp = performCalc(10, 10);
print("temp = $temp\n");

sub performCalc {
    my ($firstVar, $secondVar) = @_;
   my $square = sub {
        return($_[0] ** 2);
    };

    return(&$square($firstVar) + &$square($secondVar));
};
```

WWW.PRDEV.COM

خروجی برنامه به شکل زیر است :

```
temp = 200
```

این مثال تا حدودی جزیی است . ولی وجود توابع خصوصی در زبان برنامه نویسی Perl نشان می دهد که شما می توانید از امکانات و توانایی های این زبان در جهت دستیابی به اهداف خود در برنامه نویسی استفاده کنید .

اعمال محاسباتی و باز کردن یک فایل از عمده موارد استفاده توابع خصوصی می باشد .

در فصل بعد با فایلها در Perl آشنا خواهید شد .

WWW.PRDEV.COM

F) Perl and Files

همانطور که می دانید فایلها مجموعه ای از بایتهای اطلاعاتی می باشند که در رسانه های ذخیره سازی مانند دیسک سخت ذخیره شده اند .

مزیت استفاده از فایلها این است که پایداری آنها طولانی مدت است و مدت زمان زیادی می توان آنها را نگهداری کرد .

در این فصل شما با چگونگی کار با فایلها در زبان برنامه نویسی Perl آشنا خواهید شد .

در Perl ، ۴ عملگر اصلی کار با فایلها وجود دارد :

۱ – Open File

۲ – Read From File

۳ – Write To File

۴ – Close File

در باز کردن یک فایل ، یک اتصال بین برنامه کاربردی شما و موقعیت ذخیره سازی فایل در دیسک سخت ایجاد می شود که در بستن یک فایل این اتصال از بین می رود .

هر یک از فایلها یک نام کامل منحصر بفرد (طبق قواعد نامگذاری فایلها) دارند که نمی توانند توسط سایر فایلها مغشوش شوند .

این نام کامل منحصر بفرد شامل : نام دیسک سخت ، دایرکتوری و نام خود فایل می باشد .

شاید این سوال در ذهن شما ایجاد شود که چگونه می توان در دیسک سختمان و در چند دایرکتوری مختلف چند فایل با نامهای یکسان داشت ؟

پاسخ این است که همانطور که گفته شد ، دایرکتوری هم جزء نام کامل فایل می باشد . پس اگر نام دایرکتوریشان متفاوت باشد ، می توان چند فایل هم نام (در دایرکتوری های متفاوت داشت) .

به چند نام کامل فایل توجه کنید :

WWW.PRDEV.COM

```
c:/windows/win95.txt
c:/windows/command/scandisk.ini
c:/a_long_directory_name/a_long_subdirectory_name/a_long_file_name.doc
```

زمانی که یک برنامه شروع می شود ۳ اتصال مهم که در هر برنامه ای که ایجاد می شود وجود دارد :

```
STDIN, STDOUT, and STDERR
```

در اصل این اسامی File Handle های ما هستند .

File Handle ها متغیرهایی هستند که در استفاده و کار با فایلها استفاده می شوند .

به File Handle ها قبل از کار با فایلها نیاز داریم .

STDIN ، File Handle ای است که از ورودی های استاندارد موجود ورودی ها را می خواند مثل :

کی بورد ، اسکریپتهای نرمال Perl و سایر عناصر ورودی ما در سیستم .

SDTOUT و SDTERR ، File Handle ای است که در پورت خروجی می نویسد .

یکی از توابعی که در درسهای گذشته استفاده کردیم ، ()print بود که حال دریافتیم که این تابع برای برای SDTOUT FILE HANDLE استفاده می شود .

استفاده از SDTIN File handle :

خواندن از ورودی مانند خواندن از صفحه کلید یکی از ساده ترین و پرکاربردترین SDTIN هایی است که در Perl از آن استفاده می شود .

این کار در DOS بوسیله کلید Ctrl+Z و در UNIX بوسیله Ctrl-D صورت خواهد گرفت .

```
while (<STDIN>) {
    print();
}
```

کاراکترهای <> این اجازه را به شما می دهند که یک خط روردی را از SDTIN File Handle بخوانید .

```
while ($inputLine = <STDIN>) {
    print($inputLine);
}
```

WWW.PRDEV.COM

در این مثال ، کاراکترهای <> بجای SDTIN متغیر $_ را بعنوان ورودی گرفته اند .

متغیر $_ انواع گوناگونی از داده ها را می تواند بگیرد .

عملگرهای تست فایلها :

Table 9.1: File Operands

Operator	*Description*
-A OPERAND	Returns the access age of OPERAND when the program started.
-bOPERAND	Tests if OPERAND is a block device.
-BOPERAND	Tests if OPERAND is a binary file. If OPERAND is a file handle,
	then the current buffer is examined, instead of the file itself.
-c OPERAND	Tests if OPERAND is a character device.
-COPERAND	Returns the inode change age of OPERAND when the program started.
-dOPERAND	Tests if OPERAND is a directory.
-eOPERAND	Tests if OPERAND exists.
-fOPERAND	Tests if OPERAND is a regular file as opposed to a directory,
	symbolic link or other type of file.
-g OPERAND	Tests if OPERAND has the setgid bit set.
-kOPERAND	Tests if OPERAND has the sticky bit set.
-lOPERAND	Tests if OPERAND is a symbolic link. Under DOS,
	this operator always will return false.
-M OPERAND	Returns the age of OPERAND in days when the program started.
-oOPERAND	Tests if OPERAND is owned by the effective uid.
	Under DOS, it always returns true.
-OOPERAND	Tests if OPERAND is owned by the read uid/gid.
	Under DOS, it always returns true.
-pOPERAND	Tests if OPERAND is a named pipe.
-rOPERAND	Tests if OPERAND can be read from.
-ROPERAND	Tests if OPERAND can be read from by the real uid/gid.
	Under DOS, it is identical to -r.

WWW.PRDEV.COM

-sOPERAND	Returns the size of OPERAND in bytes.
	Therefore, it returns true if OPERAND is non-zero.
-SOPERAND	Tests if OPERAND is a socket.
-tOPERAND	Tests if OPERAND is opened to a tty.
-TOPERAND	Tests if OPERAND is a text file. If OPERAND is a file handle,
	then the current buffer is examined, instead of the file itself.
-uOPERAND	Tests if OPERAND has the setuid bit set.
-w OPERAND	Tests if OPERAND can be written to.
-W OPERAND	Tests if OPERAND can be written to by the real uid/gid.
	Under DOS, it is identical to -w.
-xOPERAND	Tests if OPERAND can be executed.
-XOPERAND	Tests if OPERAND can be executed by the real uid/gid.
	Under DOS, it is identical to -x.
-zOPERAND	Tests if OPERAND size is zero.

توابع کار با فایلها :

- binmode(FILE_HANDLE) This function puts FILE_HANDLE into a binary mode.
- chdir(DIR_NAME) Causes your program to use DIR_NAME as the current directory. It will return true if the change was successful, false if not.
- chmod(MODE, FILE_LIST) This UNIX-based function changes the permissions for a list of files. A count of the number of files whose permissions was changed is returned. There is no DOS equivalent for this function.
- chown(UID, GID, FILE_LIST) This UNIX-based function changes the owner and group for a list of files. A count of the number of files whose ownership was changed is returned. There is no DOS equivalent for this function.
- close(FILE_HANDLE) Closes the connection between your program and the file opened with FILE_HANDLE.
- closedir(DIR_HANDLE) Closes the connection between your program and the directory opened with DIR_HANDLE.
- eof(FILE_HANDLE) Returns true if the next read on FILE_HANDLE will result in hitting the end of the file or if the

WWW.PRDEV.COM

file is not open. If FILE_HANDLE is not specified the status of the last file read is returned. All input functions return the undefined value when the end of file is reached, so you'll almost never need to use eof().

- fcntl(FILE_HANDLE, Implements the fcntl() function which lets FUncTION, SCALAR) you perform various file control operations. Its use is beyond the scope of this course.
- fileno(FILE_HANDLE) Returns the file descriptor for the specified FILE_HANDLE.
- flock(FILEHANDLE, OPERATION) This function will place a lock on a file so that multiple users or programs can't simultaneously use it. The flock() function is beyond the scope of this book.
- getc(FILE_HANDLE) Reads the next character from FILE_HANDLE. If FILE_HANDLE is not specified, a character will be read from STDIN. glob(EXPRESSION) Returns a list of files that match the specification of EXPRESSION, which can contain wildcards. For instance, glob("*.pl") will return a list of all Perl program files in the current directory.
- ioctl(FILE_HANDLE, Implements the ioctl() function which lets FUncTION, SCALAR) you perform various file control operations. Its use is beyond the scope of this book. For more in-depth discussion of this function see Que's *Special Edition Using Perl for Web Programming*.
- link(OLD_FILE_NAME, This UNIX-based function creates a new NEW_FILE_NAME) file name that is linked to the old file name. It returns true for success and false for failure. There is no DOS equivalent for this function.

lstat(FILE_HANDLE_OR_Returns file statistics in a 13-element array. FILE_NAME) lstat() is identical to stat() except that it can also return information about symbolic links.

- mkdir(DIR_NAME, MODE) Creates a directory named DIR_NAME. If you try to create a subdirectory, the parent must already exist. This function returns false if the directory can't be created. The special variable $! is assigned the error message.
- open(FILE_HANDLE, EXPRESSION) Creates a link between FILE_HANDLE and a file specified by EXPRESSION.
- opendir(DIR_HANDLE, DIR_NAME) Creates a link between DIR_HANDLE and the directory specified by DIR_NAME. opendir() returns true if successful, false otherwise.
- pipe(READ_HANDLE), Opens a pair of connected pipes like the WRITE_HANDLE) corresponding system call. Its use is beyond the scope of this book. For more on this function see Que's *Special Edition Using Perl for Web Programming*. print FILE_HANDLE (LIST) Sends a list of strings to FILE_HANDLE. If FILE_HANDLE is not specified, then STDOUT is used.

www.pbookshop.com

- printf FILE_HANDLESends a list of strings in a format specified by (FORMAT, LIST) FORMAT to FILE_HANDLE. If FILE_HANDLE is not specified, then STDOUT is used.
- read(FILE_HANDLE, BUFFER, Reads bytes from FILE_HANDLE starting at LENGTH,LENGTHOFFSET) OFFSET position in the file into the scalar variable called BUFFER. It returns the number of bytes read or the undefined value.
- readdir(DIR_HANDLE) Returns the next directory entry from DIR_HANDLE when used in a scalar context. If used in an array context, all of the file entries in DIR_HANDLE will be returned in a list. If there are no more entries to return, the undefined value or a null list will be returned depending on the context.
- readlink(EXPRESSION) This UNIX-based function returns that value of a symbolic link. If an error occurs, the undefined value is returned and the special variable $! is assigned the error message. The $_ special variable is used if EXPRESSION is not specified.
- rename(OLD_FILE_NAME, Changes the name of a file. You can use this NEW_FILE_NAME) function to change the directory where a file resides, but not the disk drive or volume.
- rewinddir(DIR_HANDLE) Resets DIR_HANDLE so that the next readdir() starts at the beginning of the directory.
- rmdir(DIR_NAME) Deletes an empty directory. If the directory can be deleted it returns false and $! is assigned the error message. The $ special variable is used if DIR_NAME is not specified.
- seek(FILE_HANDLE, POSITION, Moves to POSITION in the file connected to WHEncE) FILE_HANDLE. The WHEncE parameter determines if POSITION is an offset from the beginning of the file (WHEncE=0), the current position in the file (WHEncE=1), or the end of the file (WHEncE=2).
- seekdir(DIR_HANDLE, POSITION) Sets the current position for readdir(). POSITION must be a value returned by the telldir() function.
- select(FILE_HANDLE) Sets the default FILE_HANDLE for the write() and print() functions. It returns the currently selected file handle so that you may restore it if needed.
- sprintf(FORMAT, LIST) Returns a string whose format is specified by FORMAT.
- stat(FILE_HANDLE_OR_Returns file statistics in a 13-element array. FILE_NAME)
- symlink(OLD_FILE_NAME, This UNIX-based function creates a new NEW_FILE_NAME) file name symbolically linked to

www.PREY.com

the old file name. It returns false if the NEW_FILE_NAME cannot be created.

- sysread(FILE_HANDLE, BUFFER, Reads LENGTH bytes from FILE_HANDLE starting LENGTH,OFFSET) at OFFSET position in the file into the scalar variable called BUFFER. It returns the number of bytes read or the undefined value.
- syswrite(FILE_HANDLE, BUFFER, Writes LENGTH bytes from FILE_HANDLE starting LENGTH, OFFSET) at OFFSET position in the file into the scalar variable called BUFFER. It returns the number of bytes written or the undefined value.
- tell(FILE_HANDLE) Returns the current file position for FILE_HANDLE. If FILE_HANDLE is not specified, the file position for the last file read is returned.
- telldir(DIR_HANDLE) Returns the current position for DIR_HANDLE. The return value may be passed to seekdir() to access a particular location in a directory.
- truncate(FILE_HANDLE, LENGTH) Truncates the file opened on FILE_HANDLE to be LENGTH bytes long.
- unlink(FILE_LIST) Deletes a list of files. If FILE_LIST is not specified, then $ will be used. It returns the number of files successfully deleted. Therefore, it returns false or 0 if no files were deleted.
- utime(FILE_LIST) This UNIX-based function changes the access and modification times on each file in FILE_LIST.
- write(FILE_HANDLE) Writes a formatted record to FILE_HANDLE.

WWW.IRDEO.COM