



Perl Tutorial – Learn Perl With Examples

[Read](#)[Practice](#)

[Perl](#) - это универсальный, высокоуровневый интерпретируемый и динамический язык программирования. На начальном уровне Perl был разработан только для управления системой и обработки текста, но в более поздних версиях Perl получил возможность обрабатывать регулярные выражения, сетевые сокеты и т.д. В настоящее время Perl популярен благодаря своей способности обрабатывать регулярные выражения. Первой версией Perl была 1.0, выпущенная 18 декабря 1987 года. Perl 6 отличается от Perl 5 тем, что это полностью объектно-ориентированная повторная реализация Perl 5.



Темы:

- [Концепции OOPs](#)
 - [Класс](#)
 - [Объект](#)
 - [Метод](#)
 - [Полиморфизм](#)
 - [Наследование](#)
 - [Инкапсуляция](#)



- [Основные функции и приложения](#)
- [Загрузка и установка Perl](#)
- [Запустите свою первую программу](#)
- [Основы Perl](#)
 - [Переменные](#)
 - [Операторы](#)
 - [Числа](#)
- [Типы данных](#)
 - [Скаляры](#)
 - [Массивы](#)
 - [Хэши](#)
 - [Строки](#)
- [Принятие решений](#)
- [Циклы](#)
 - [для цикла](#)
 - [цикл foreach](#)
 - [цикл while и do ...while](#)
- [Абстракция](#)
- [Подпрограммы](#)
- [Модули и пакеты](#)
- [Регулярное выражение](#)
- [Обработка файлов](#)
 - [Чтение и запись](#)
 - [Основные операции с файлами](#)
 - [Работа с файлами Excel](#)
- [Обработка ошибок](#)

Основные функции

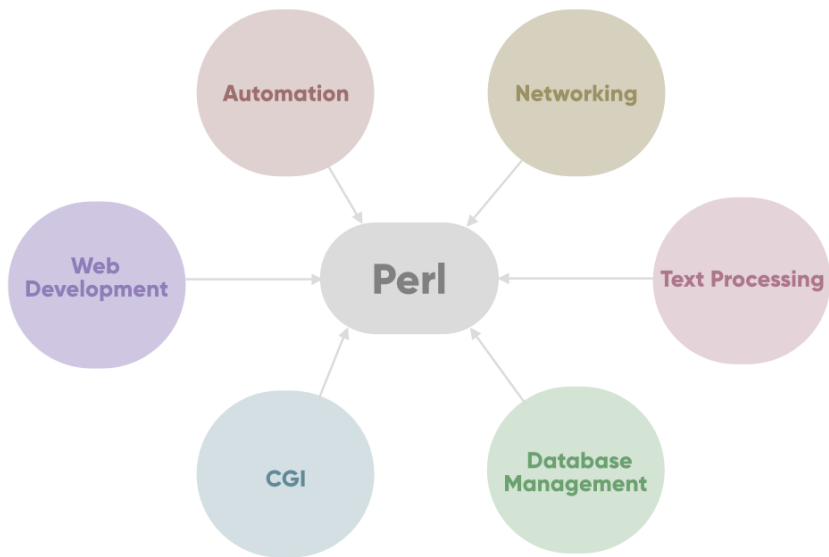
[УPerl](#) есть много причин, по которым он популярен и востребован. Ниже перечислены некоторые из причин:

- **Perl [Легко начать](#):** это язык высокого уровня, поэтому он ближе к другим популярным языкам программирования, таким как [C](#), [C++](#), и, таким образом, становится простым в изучении для любого.
- **Обработка текста:** Поскольку аббревиатура “Язык практического извлечения и составления отчетов” предполагает, что Perl обладает высокими возможностями обработки текста, с помощью

которых он может легко генерировать отчеты из различных текстовых файлов.

- **Содержит лучшие функции:** Perl содержит функции различных языков, таких как `C`, `sed`, `awk`, `sh` и др. что делает Perl более полезным и продуктивным.
- **Системное администрирование:** Perl очень упрощает задачу системного администрирования. Вместо того, чтобы зависеть от множества языков, просто используйте Perl для выполнения всей задачи системного администрирования.
- **Веб и Perl:** Perl может быть встроен в веб-серверы для увеличения вычислительной мощности, и у него есть пакет DBI, который очень упрощает интеграцию с веб-базой данных.

Области применения



Почему и почему не стоит использовать Perl?

ЗАЧЕМ ИСПОЛЬЗОВАТЬ PERL?	ЧТО НЕ ТАК С PERL?
Perl обеспечивает поддержку кроссплатформенности и совместим с языками разметки, такими как HTML, XML и т.д.	Perl не поддерживает переносимость из-за модулей CPAN.
Это бесплатное программное обеспечение с открытым исходным кодом, которое лицензируется по	Программы выполняются медленно, и их необходимо интерпретировать каждый раз,

Artistic и GNU General Public License (GPL).	когда вносятся какие-либо изменения.
Это встраиваемый язык, поэтому его можно встраивать в веб-серверы и серверы баз данных.	В Perl один и тот же результат может быть достигнут несколькими различными способами, которые делают код неопрятным, а также нечитаемым.
Он очень эффективен при работе с текстом, т. е. с регулярными выражениями. Он также предоставляет возможность сокета.	Коэффициент удобства использования ниже по сравнению с другими языками.

Начало работы с Perl

Поскольку Perl синтаксически во многом похож на другие широко используемые языки, на Perl легче программировать и осваивать. Программы на Perl можно писать в любом текстовом редакторе, таком как notepad, notepad ++ или что-либо в этом роде. Также можно использовать [интерактивную среду разработки для написания Perl-кодов](#) или даже установить ее в своей системе, чтобы сделать написание этих кодов более выполнимым, поскольку IDE предоставляют множество функций, таких как интуитивно понятный редактор кода, отладчик, компилятор и т.д.

Для начала написания кодов Perl и выполнения различных интересных и полезных операций в системе пользователя должен быть установлен Perl. Это можно сделать, следуя пошаговым инструкциям, приведенным ниже:

Что, если Perl уже существует?? Давайте проверим!!!

В настоящее время для выполнения своих операций многим программным приложениям требуется Perl, следовательно, версия Perl может быть включена в установочный пакет программного обеспечения. Во многих системах Linux предустановлен Perl, также Macintosh предоставляет предустановленный Perl в своих системах.

Чтобы проверить, установлен ли на вашем устройстве Perl или нет, просто перейдите в **командную строку** (для **Windows** найдите **cmd** в диалоговом окне Запуска (**Windows** + R), для **Linux** откройте терминал с помощью **Ctrl+Alt+T**, для **macOS** используйте **Control+Option+Shift+T**)
Теперь выполните следующую команду:

```
perl -v
```

Если Perl уже установлен, он сгенерирует сообщение со всеми подробностями о доступной версии Perl, в противном случае, если Perl не установлен, возникнет ошибка с указанием *неверной команды или имени файла*

```
C:\Users\GeeksForGeeks>perl -v

This is perl 5, version 26, subversion 3 (v5.26.3) built for MSWin32-x64-multi-thread
(with 2 registered patches, see perl -V for more detail)

Copyright 1987-2018, Larry Wall

Binary build 2603 [a95bce075] provided by ActiveState http://www.ActiveState.com
Built Dec 17 2018 09:46:45

Perl may be copied only under the terms of either the Artistic License or the
GNU General Public License, which may be found in the Perl 5 source kit.







Complete documentation for Perl, including FAQ lists, should be found on
this system using "man perl" or "perldoc perl". If you have access to the
Internet, point your browser at http://www.perl.org/, the Perl Home Page.

C:\Users\GeeksForGeeks>
```

Загрузка и установка Perl:

Прежде чем приступить к процессу установки, вам необходимо загрузить его. Для этого все версии Perl для Windows, Linux и macOS доступны на perl.org.

We recommend that you always run the latest stable version, currently 5.30.0. If you're running a version older than 5.8.3, you may find that the latest version of CPAN modules will not work.

Unix/Linux	macOS	Windows
 Included (may not be latest)	 Included (may not be latest)	 Strawberry Perl & ActiveState Perl
 GET STARTED	 GET STARTED	 GET STARTED

Загрузите Perl и следуйте дальнейшим инструкциям по установке Perl.

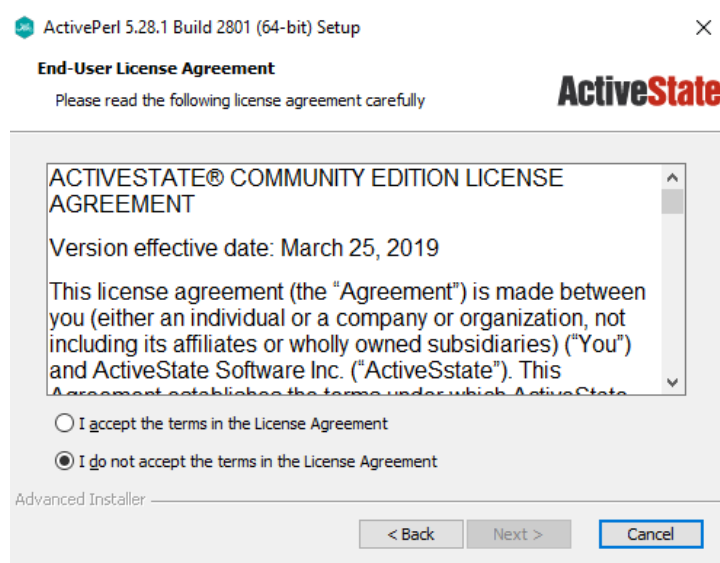
Начинаем с установки:

Windows

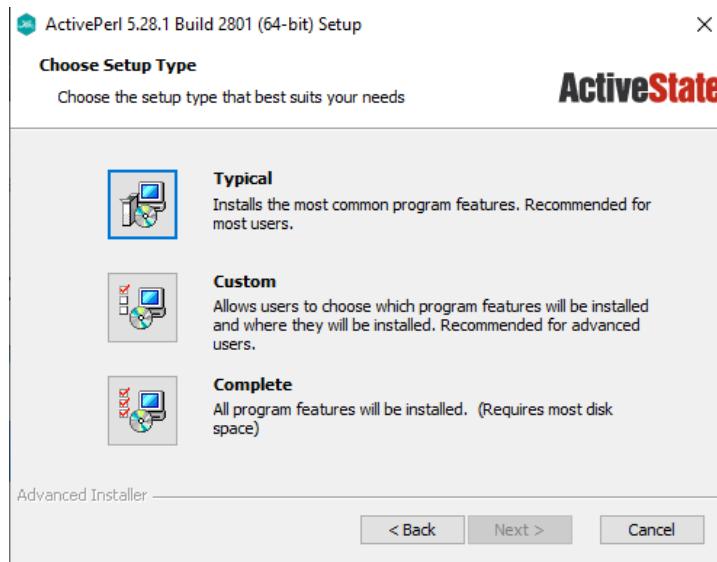
- Приступая к работе:



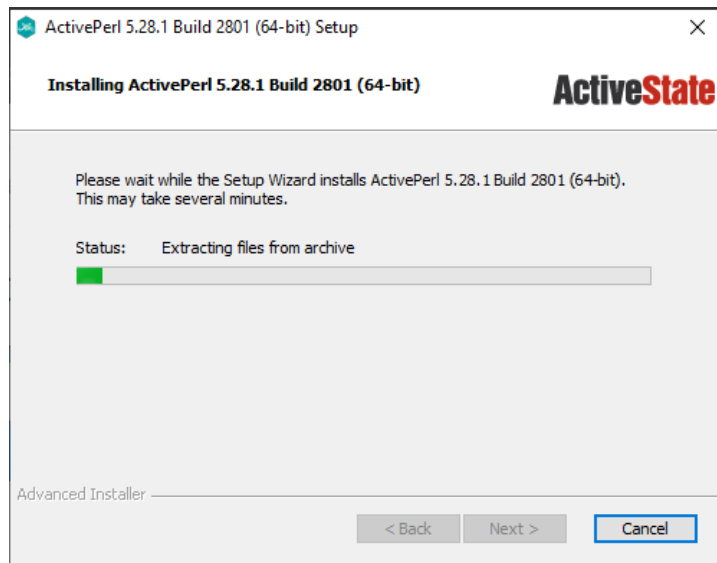
- Ознакомление с пользовательским лицензионным соглашением:



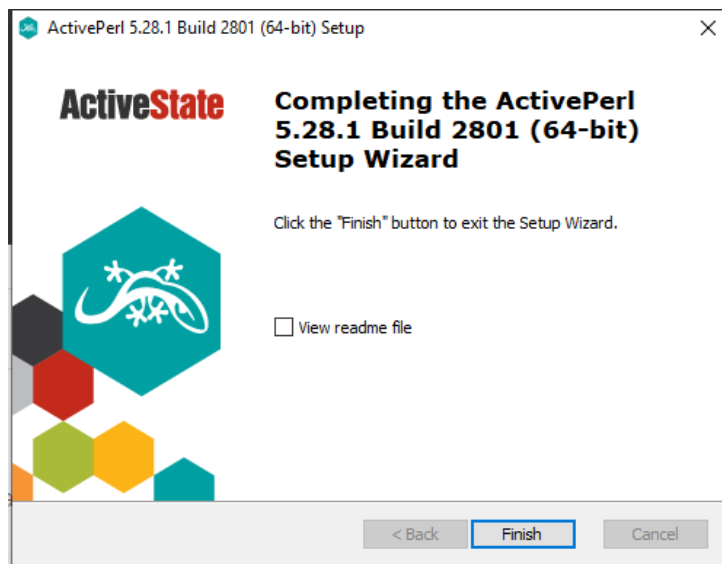
- Выбор того, что установить:



- Процесс установки:



- Закончена установка:



Linux

- Changing Directory to install Perl:

```
divya@divya-H110MHC: ~/Downloads/per/ActivePerl-5.28.1.2801-x86_64-linux-glibc-2.12-cb0...
File Edit View Search Terminal Help
divya@divya-H110MHC:~$ cd Downloads/
divya@divya-H110MHC:~/Downloads$ cd per
divya@divya-H110MHC:~/Downloads/per$ cd ActivePerl-5.28.1.2801-x86_64-linux-glibc-2.12-cb03d3e0/
divya@divya-H110MHC:~/Downloads/per/ActivePerl-5.28.1.2801-x86_64-linux-glibc-2.12-cb03d3e0$ ls
install.sh perl README.txt support
divya@divya-H110MHC:~/Downloads/per/ActivePerl-5.28.1.2801-x86_64-linux-glibc-2.12-cb03d3e0$
```

• Starting the Installation Process:

```
divya@divya-H110MHC: ~/Downloads/per/ActivePerl-5.28.1.2801-x86_64-linux-glibc-2.12-cb0...
File Edit View Search Terminal Help
divya@divya-H110MHC:~$ cd Downloads/
divya@divya-H110MHC:~/Downloads$ cd per
divya@divya-H110MHC:~/Downloads/per$ cd ActivePerl-5.28.1.2801-x86_64-linux-glibc-2.12-cb03d3e0/
divya@divya-H110MHC:~/Downloads/per/ActivePerl-5.28.1.2801-x86_64-linux-glibc-2.12-cb03d3e0$ ls
install.sh perl README.txt support
divya@divya-H110MHC:~/Downloads/per/ActivePerl-5.28.1.2801-x86_64-linux-glibc-2.12-cb03d3e0$ sh install.sh
Checking package...done

Welcome to ActivePerl

This installer can install ActivePerl in any location of your choice. You do not need root privileges. However, please make sure that you have write access to this location.

Enter top level directory for install? [/opt/ActivePerl-5.28] 
```

• Choosing the Directory to Install Perl:

```
divya@divya-H110MHC: ~/Downloads/per/ActivePerl-5.28.1.2801-x86_64-linux-glibc-2.12-cb0...
File Edit View Search Terminal Help
divya@divya-H110MHC:~/Downloads/per$ cd ActivePerl-5.28.1.2801-x86_64-linux-glibc-2.12-cb03d3e0/
divya@divya-H110MHC:~/Downloads/per/ActivePerl-5.28.1.2801-x86_64-linux-glibc-2.12-cb03d3e0$ ls
install.sh perl README.txt support
divya@divya-H110MHC:~/Downloads/per/ActivePerl-5.28.1.2801-x86_64-linux-glibc-2.12-cb03d3e0$ sh install.sh
Checking package...done

Welcome to ActivePerl

This installer can install ActivePerl in any location of your choice. You do not need root privileges. However, please make sure that you have write access to this location.

Enter top level directory for install? [/opt/ActivePerl-5.28]

The ActivePerl documentation is available in HTML format. If installed it will be available from file:///opt/ActivePerl-5.28/html/index.html. If not installed you will still be able to read all the basic perl and module documentation using the man or perldoc utilities.

Install HTML documentation [yes] 
```

• Finishing the Installation:


```
divya@divya-H110MHC: ~/Downloads/per/ActivePerl-5.28.1.2801-x86_64-linux-glibc-2.12-cb0...
File Edit View Search Terminal Help

Proceed? [yes] yes
ok.

Installing ActivePerl...
Copying files to /opt/ActivePerl-5.28...
Install of ActivePerl to /opt/ActivePerl-5.28 failed:

    mkdir /opt/ActivePerl-5.28: Permission denied at perl/lib/perl5/site_perl/5.
28.1/ActiveState/RelocateTree.pm line 388.

Sorry about the inconvenience. If this looks like a problem with
the installer please report it to the ActivePerl bug database,
available online at:

    http://bugs.ActiveState.com/ActivePerl/

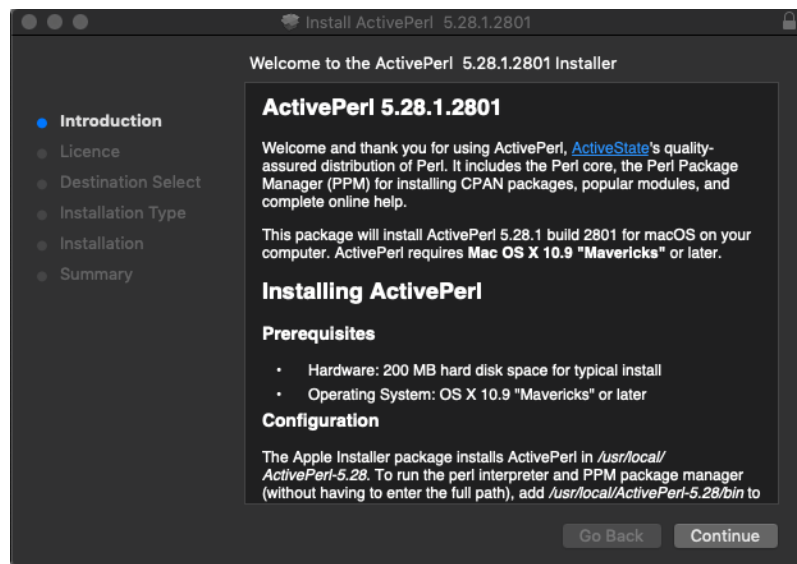
For general questions or comments about ActivePerl, please contact
us at <support@activestate.com>.

Thank you for trying to install ActivePerl!

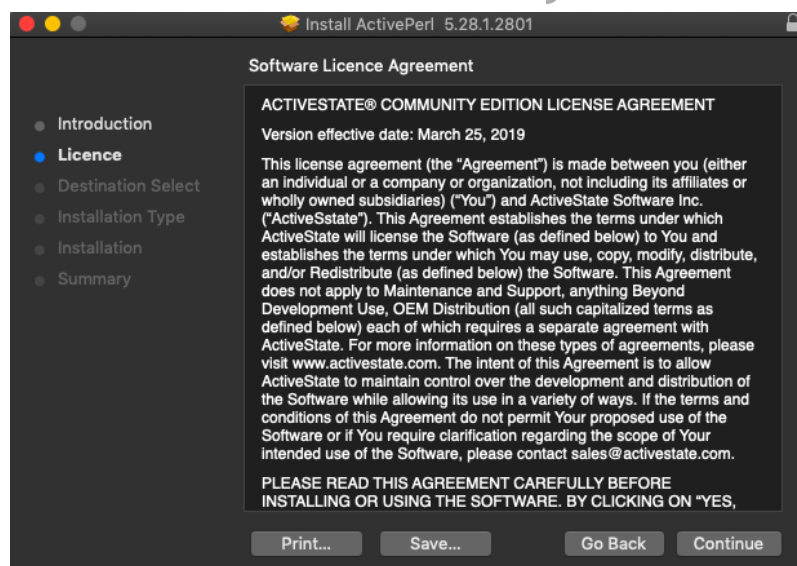
divya@divya-H110MHC:~/Downloads/per/ActivePerl-5.28.1.2801-x86_64-linux-glibc-2.
12-cb03d3e0$
```

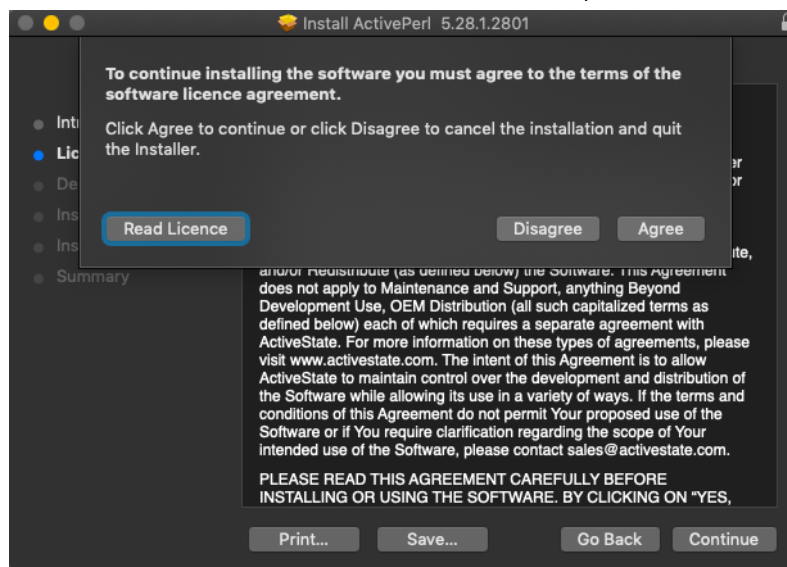
MacOS

• Getting Started:

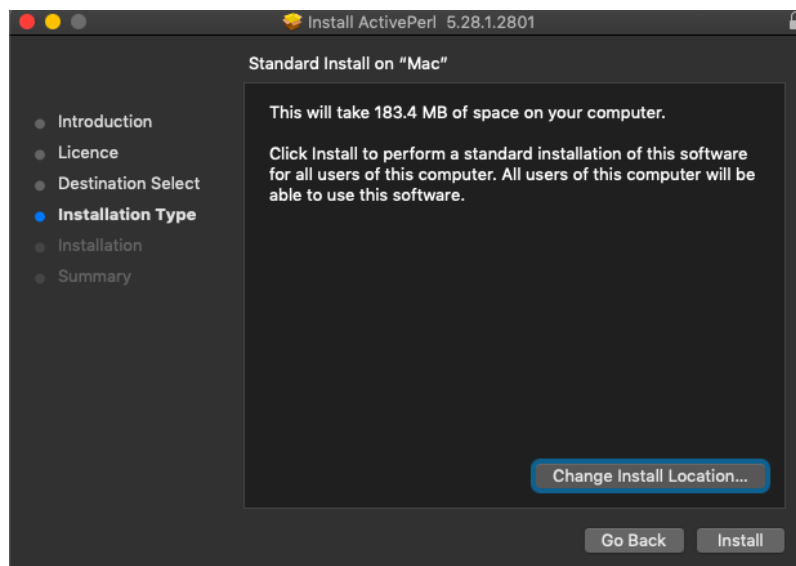


• Getting done with the User's License Agreement:

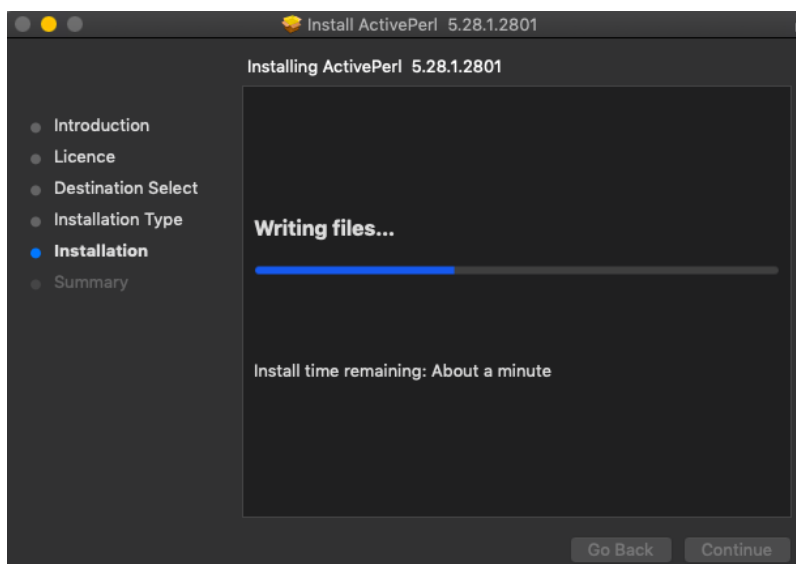




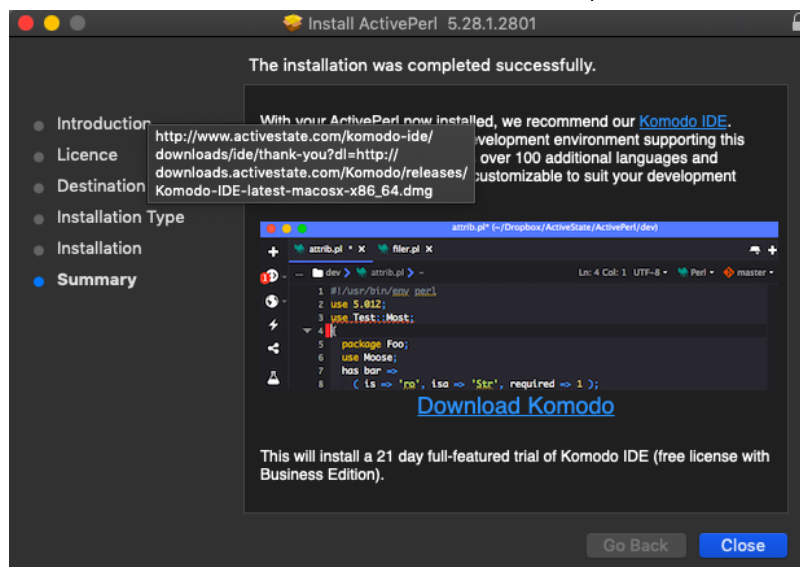
- Choosing Installation Location:



- Installation Process:



- Finishing Installation:



Как запустить программу на Perl?

Давайте рассмотрим простую программу Hello World.

```
#!/usr/bin/perl

# Modules used
use strict;
use warnings;

# Print function
print("Hello World\n");
```

Как правило, есть два способа запустить программу на Perl-

- **Использование онлайн-IDE:** Вы можете использовать различные онлайн-IDE, которые можно использовать для запуска программ на Perl без установки.
- **Использование командной строки:** Вы также можете использовать параметры командной строки для запуска программы на Perl. Приведенные ниже шаги демонстрируют, как запустить программу на Perl из командной строки в операционных системах Windows / Unix:

Windows

Откройте командную строку, а затем для компиляции введите код `perl HelloWorld.pl`. Если в вашем коде нет ошибок, то он будет выполнен правильно и будут отображены выходные данные.

```
C:\Users\GeeksForGeeks>perl HelloWorld.pl
Hello World

C:\Users\GeeksForGeeks>
```

Unix / Linux

Откройте терминал вашей ОС Unix / Linux и затем для компиляции введите код **perl hello.pl** . Если в вашем коде нет ошибок, то он будет выполнен правильно и будут отображены выходные данные .

```
anshu@geeksforgeeks:~$ perl hello.pl
Hello World
anshu@geeksforgeeks:~$
```

Основы Perl

Переменные

Переменные - это определяемые пользователем слова, которые используются для хранения значений, передаваемых программе, которые будут использоваться для оценки кода. Каждая программа Perl содержит значения, с которыми код выполняет свои операции. Этими значениями нельзя манипулировать или сохранять их без использования переменной. Значение можно обработать, только если оно сохранено в переменной, используя имя переменной.

Значение - это данные, передаваемые программе для выполнения

операций манипулирования. Эти данные могут быть числами, строками, символами, списками и т.д.

Пример:

Values:

5

geeks

15

Variables:

\$a = 5;

\$b = "geeks";

\$c = 15;

Операторы

Операторы являются основным строительным блоком любого языка программирования. Операторы позволяют программисту выполнять различные виды операций над операндами. Эти операторы могут быть классифицированы в зависимости от их различной функциональности:

- [Арифметические операторы](#)
- [Операторы отношения](#)
- [Логические операторы](#)
- [Побитовые операторы](#)
- [Операторы присваивания](#)
- [Тернарный оператор](#)

```
# Perl Program to illustrate the Operators
```

```
# Operands
```

```
$a = 10;
```

```
$b = 4;
```

```
$c = true;
```

```
$d = false;
```

```
# using arithmetic operators
```

```
print "Addition is: ", $a + $b, "\n";
```

```
print "Subtraction is: ", $a - $b, "\n" ;
```

```
# using Relational Operators
```

```
if ($a == $b)
```

```
{
```

```
    print "Equal To Operator is True\n";
```

```
}  
else  
{  
    print "Equal To Operator is False\n";  
}  
  
# using Logical Operator 'AND'  
$result = $a && $b;  
print "AND Operator: ", $result, "\n";  
  
# using Bitwise AND Operator  
$result = $a & $b;  
print "Bitwise AND: ", $result, "\n";  
  
# using Assignment Operators  
print "Addition Assignment Operator: ", $a += $b, "\n";
```

Вывод:

```
Addition is: 14  
Subtraction is: 6  
Equal To Operator is False  
AND Operator: 4  
Bitwise AND: 0  
Addition Assignment Operator: 14
```

Число и его типы

Число в Perl - это математический объект, используемый для подсчета, измерения и выполнения различных математических операций. Условный символ, представляющий число, называется цифрой. Эти цифры, помимо их использования в математических операциях, также используются для упорядочивания (в виде серийных номеров).

Типы чисел:

- Целые числа
- Плавающие числа
- Шестнадцатеричные числа
- Восьмеричные числа
- Двоичные числа

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

```
# Perl program to illustrate
# the use of numbers

# Integer
$a = 20;

# Floating Number
$b = 20.5647;

# Scientific value
$c = 123.5e-10;

# Hexadecimal Number
$d = 0xc;

# Octal Number
$e = 074;

# Binary Number
$f = 0b1010;

# Printing these values
print("Integer: ", $a, "\n");
print("Float Number: ", $b, "\n");
print("Scientific Number: ", $c, "\n");
print("Hex Number: ", $d, "\n");
print("Octal number: ", $e, "\n");
print("Binary Number: ", $f, "\n");
```

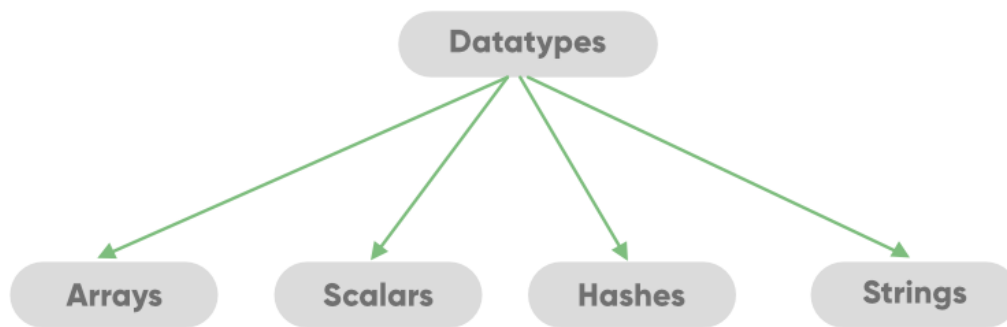
Вывод:

```
Integer: 20
Float Number: 20.5647
Scientific Number: 1.235e-08
Hex Number: 12
Octal number: 60
Binary Number: 10
```

Чтобы узнать больше о числах, пожалуйста, обратитесь к [Числам в Perl](#)

Типы данных

Типы данных определяют тип данных, которые может содержать допустимая переменная [Perl](#). Perl - это свободно типизированный язык. Нет необходимости указывать тип данных при их использовании в программе Perl. Интерпретатор Perl выберет тип на основе контекста самих данных.



Скаляры

Это отдельная единица данных, которая может быть целым числом, запятой с плавающей запятой, символом, строкой, абзацем или целой веб-страницей.

Пример:

```
# Perl program to demonstrate
# scalars variables

# a string scalar
$name = "Alex";

# Integer Scalar
$rollno = 13;

# a floating point scalar
$percentage = 87.65;

# In hexadecimal form
$hexadec = 0xcd;

# Alphanumeric String
$alphanumeric = "gfg21";

# special character in string scalar
$specialstring = "^gfg";

# to display the result
print "Name = $name\n";
print "Roll number = $rollno\n";
print "Percentage = $percentage\n";
print "Hexadecimal Form = $hexadec\n";
print "String with alphanumeric values = $alphanumeric\n";
print "String with special characters = $specialstring\n";
```


Вывод:

```
Name = Alex
Roll number = 13
Percentage = 87.65
Hexadecimal Form = 205
String with alphanumeric values = gfg21
String with special characters = ^gfg
```

Чтобы узнать больше о скалярах, пожалуйста, обратитесь к

Скалярам в Perl.

Массивы

Массив - это переменная, которая хранит значение того же типа данных в виде списка. Чтобы объявить массив в Perl, мы используем знак '@' перед именем переменной.

```
@number = (40, 55, 63, 17, 22, 68, 89, 97, 89)
```

40	55	63	17	22	68	89	97	89
0	1	2	3	4	5	6	7	8

<- Array Indices

Array Length = 9

First Index = 0

Last Index = 8

Это создаст массив целых чисел, содержащий значения 40, 55, 63, 17 и многие другие. Чтобы получить доступ к одному элементу массива, мы используем знак '\$'.

```
$number[0]
```

На выходе будет 40.

Создание массива и доступ к элементам:

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

```
# Perl Program for array creation
# and accessing its elements

# Define an array
@arr1 = (1, 2, 3, 4, 5);

# using qw function
@arr2 = qw /This is a Perl Tutorial by GeeksforGeeks/;

# Accessing array elements
print "Elements of arr1 are:\n";
print "$arr1[0]\n";
print "$arr1[3]\n";

# Accessing array elements
# with negative index
print "\nElements of arr2 are:\n";
print "$arr2[-1]\n";
print "$arr2[-3]\n";
```

Вывод:

Elements of arr1 are:

1
4

Elements of arr2 are:

GeeksforGeeks
Tutorial

Перебираем массив:

```
#!/usr/bin/perl

# Perl program to illustrate
# iteration through an array

# array creation
@arr = (11, 22, 33, 44, 55);

# Iterating through the range
print("Iterating through range:\n");

# size of array
$len = @arr;

for ($b = 0; $b < $len; $b = $b + 1)
{
```

```
print "\@arr[$b] = $arr[$b]\n";
}

# Iterating through loops
print("\nIterating through loops:\n");

# foreach loop
foreach $a (@arr)
{
    print "$a ";
}
```

Вывод:

Iterating through range:

```
@arr[0] = 11
@arr[1] = 22
@arr[2] = 33
@arr[3] = 44
@arr[4] = 55
```

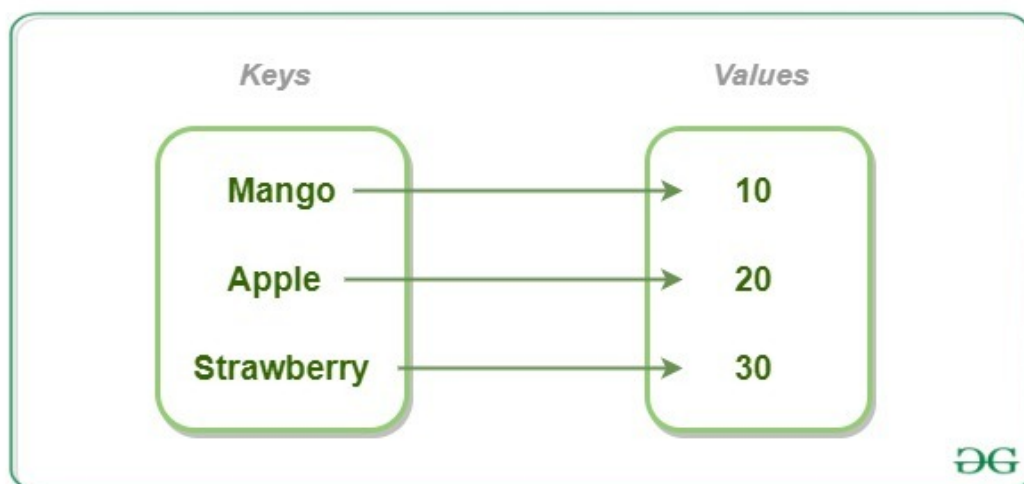
Iterating through loops:

```
11 22 33 44 55
```

Чтобы узнать больше о массивах, пожалуйста, обратитесь к [Массивы в Perl](#)

Хэши (ассоциативные массивы)

Это набор пар ключ-значение. Его также называют ассоциативными массивами. Чтобы объявить хэш в Perl, мы используем знак '%'. Для доступа к определенному значению мы используем символ '\$', за которым следует ключ в фигурных скобках.



Создание хэш-элементов и доступ к ним:

```
#!/usr/bin/perl

# Perl Program for Hash creation
# and accessing its elements

# Initializing Hash by
# directly assigning values
$Fruit{'Mango'} = 10;
$Fruit{'Apple'} = 20;
$Fruit{'Strawberry'} = 30;

# printing elements of Hash
print "Printing values of Hash:\n";
print "$Fruit{'Mango'}\n";
print "$Fruit{'Apple'}\n";
print "$Fruit{'Strawberry'}\n";

# Initializing Hash using '=>'
%Fruit2 = ('Mango' => 45, 'Apple' => 42, 'Strawberry' => 35);

# printing elements of Fruit2
print "\nPrinting values of Hash:\n";
print "$Fruit2{'Mango'}\n";
print "$Fruit2{'Apple'}\n";
print "$Fruit2{'Strawberry'}\n";
```

Вывод:

Printing values of Hash:

10
20
30

Printing values of Hash:

45
42
35

Чтобы узнать больше о хэшах, пожалуйста, обратитесь к [Хэши в Perl](#)

Строки

Строка в Perl является скалярной переменной и начинается со знака (\$) и может содержать алфавиты, цифры, специальные

символы. Строка может состоять из одного слова, группы слов или многострочного абзаца. Строка определяется пользователем в одинарных (') или двойных кавычках (").

```
#!/usr/bin/perl

# An array of integers from 1 to 10
@list = (1..10);

# Non-interpolated string
$string1 = 'Using Single quotes: @list';

# Interpolated string
$string2 = "Using Double-quotes: @list";
print("$string1\n$string2");
```

Вывод:

```
Using Single quotes: @list
Using Double-quotes: 1 2 3 4 5 6 7 8 9 10
```

Использование Escape-символа в строках:

Интерполяция создает проблему для строк, содержащих символы, которые могут стать бесполезными после интерполяции. Например, когда адрес электронной почты должен храниться в строке, заключенной в двойные кавычки, знак 'at' (@) автоматически интерполируется и принимается за начало имени массива и заменяется им. Чтобы преодолеть эту ситуацию, используется escape-символ, то есть обратная косая черта (\). Обратная косая черта вставляется непосредственно перед символом '@', как показано ниже:

```
#!/usr/bin/perl

# Assigning a variable with an email
# address using double-quotes

# String without an escape sequence
$email = "GeeksforGeeks0402@gmail.com";

# Printing the interpolated string
print("$email\n");

# Using ' ' to escape the
```

```
# interpolation of '@'
$email = "GeeksforGeeks0402@gmail.com";

# Printing the interpolated string
print($email);
```

Output:

```
GeeksforGeeks0402.com
GeeksforGeeks0402@gmail.com
```

Escaping the escape character:

Обратная косая черта является управляющим символом и используется для использования управляющих последовательностей. Когда возникает необходимость вставить управляющий символ в интерполированную строку, используется та же обратная косая черта, чтобы избежать замены управляющего символа на " (пробел). Это позволяет использовать управляющие символы в интерполированной строке.

```
#!/usr/bin/perl

# Using Two escape characters to avoid
# the substitution of escape(\) with blank
$string1 = "Using the escape(\\) character";

# Printing the Interpolated string
print($string1);
```

Вывод:

```
Using the escape(\) character
```

Чтобы узнать больше о строках, пожалуйста, обратитесь к [Строкам в Perl](#)

Поток управления

Принятие решений

Процесс принятия решений в программировании аналогичен процессу принятия решений в реальной жизни. Язык программирования использует управляющие операторы для управления ходом выполнения программы на основе определенных условий. Они используются для

ускорения и ветвления потока выполнения в зависимости от изменений состояния программы.

Инструкции по принятию решений на Perl :

- Если
- If - else
- Вложенный - If
- лестница if - elsif
- Если не
- Если только - else
- Если только - elsif

Пример 1: Для иллюстрации использования if и if-else

```
#!/usr/bin/perl

# Perl program to illustrate
# Decision-Making statements

$a = 10;
$b = 15;

# if condition to check
# for even number
if($a % 2 == 0 )
{
    printf "Even Number";
}

# if-else condition to check
# for even number or odd number
if($b % 2 == 0 )
{
    printf "\nEven Number";
}
else
{
    printf "\nOdd Number";
}
```

Вывод:

Even Number
Odd Number

Пример 2: Для иллюстрации использования вложенного-if

```
#!/usr/bin/perl

# Perl program to illustrate
# Nested if statement

$a = 10;

if($a % 2 == 0)
```

Курсы с возвратом 90% Сейчас в тренде Структуры данных и алгоритмы Базовые курсы Наука о дэ

```
    # Will only be executed
    # if above if statement
    # is true
    if($a % 5 == 0)
    {
        printf "Number is divisible by 2 and 5\n";
    }
}
```

Вывод:

```
Number is divisible by 2 and 5
```

Чтобы узнать больше о процессе принятия решений, пожалуйста, обратитесь к [Процессу принятия решений в Perl](#)

Циклы

Циклирование в языках программирования - это функция, которая облегчает многократное выполнение набора инструкций или функций, пока какое-либо условие принимает значение true. Циклы упрощают задачу программиста. Perl предоставляет различные типы циклов для обработки ситуаций в программе на основе условий. Циклы в Perl следующие :

- [для цикла](#)

```
#!/usr/bin/perl

# Perl program to illustrate
# the use of for Loop

# for loop
print("For Loop:\n");
for ($count = 1 ; $count <= 3 ; $count++)
{
    print "GeeksForGeeks\n"
}
```


Вывод:

```
For Loop:
GeeksForGeeks
GeeksForGeeks
GeeksForGeeks
```

- цикл foreach

```
#!/usr/bin/perl

# Perl program to illustrate
# the use of foreach Loop

# Array
@data = ('GEEKS', 4, 'GEEKS');

# foreach loop
print("For-each Loop:\n");
foreach $word (@data)
{
    print (" $word ");
}
```

Вывод:

```
For-each Loop:
GEEKS 4 GEEKS
```

- while и выполняйте цикл while

```
#!/usr/bin/perl

# Perl program to illustrate
# the use of foreach Loop

# while loop
$count = 3;

print("While Loop:\n");
while ($count >= 0)
{
    $count = $count - 1;
    print "GeeksForGeeks\n";
}

print("\ndo...while Loop:\n");
$a = 10;
```

```
# do..While loop
do {

    print "$a ";
    $a = $a - 1;
} while ($a > 0);
```

Вывод:

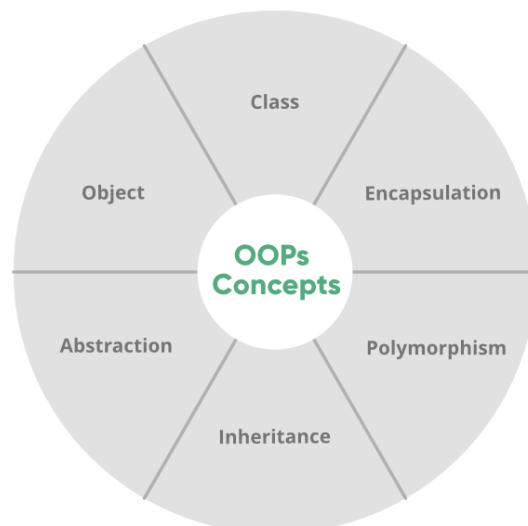
```
While Loop:
GeeksForGeeks
GeeksForGeeks
GeeksForGeeks
GeeksForGeeks
```

```
do...while Loop:
10 9 8 7 6 5 4 3 2 1
```

Чтобы узнать больше о циклах, пожалуйста, обратитесь к [Циклам в Perl](#)

Объектно-ориентированное программирование

Объектно-ориентированное программирование направлено на реализацию объектов реального мира, таких как наследование, скрытие, полиморфизм и т.д. В программировании. Основная цель ООП - связать воедино данные и функции, которые с ними работают, так, чтобы никакая другая часть кода не могла получить доступ к этим данным, кроме этой функции.



Создание класса и объекта:

```
#!/usr/bin/perl

# Perl Program for creation of a
# Class and its object
use strict;
use warnings;

package student;

# constructor
sub student_data
{
    # shift will take package name 'student'
    # and assign it to variable 'class'
    my $class_name = shift;
    my $self = {
        'StudentFirstName' => shift,
        'StudentLastName' => shift
    };
    # Using bless function
    bless $self, $class_name;

    # returning object from constructor
    return $self;
}

# Object creating and constructor calling
my $Data = student_data student("Geeks", "forGeeks");

# Printing the data
print "$Data->{'StudentFirstName'}\n";
print "$Data->{'StudentLastName'}\n";
```

Вывод:

```
Geeks
forGeeks
```

Методы: -

Методы - это сущности, которые используются для доступа к данным объекта и их изменения. Метод - это набор инструкций, которые выполняют некоторую конкретную задачу и возвращают результат вызывающей стороне. Метод может выполнять какую-то конкретную

задачу, ничего не возвращая. Методы **экономят время** и помогают нам **повторно** использовать код без повторного ввода кода.

```
sub Student_data
{
    my $self = shift;

    # Calculating the result
    my $result = $self->{'Marks_obtained'} /
                $self->{'Total_marks'};

    print "Marks scored by the student are: $result";
}
```

The above-given method can be called again and again wherever required, without doing the effort of retyping the code.

To learn more about Methods, please refer to [Methods in Perl](#)

Polymorphism:-

Полиморфизм относится к способности языков программирования OOPs эффективно различать сущности с одинаковыми именами. Это делается Perl с помощью подписи и объявления этих сущностей.

Полиморфизм лучше всего объяснить с помощью следующего примера:

```
use warnings;

# Creating class using package
package A;
sub poly_example
{
    print("This corresponds to class A\n");
}

package B;
sub poly_example
{
    print("This corresponds to class B\n");
}

B->poly_example();
A->poly_example();
```

Вывод:



```
Command Prompt
C:\Users\GeeksForGeeks>perl poly.pl
This corresponds to class B
This corresponds to class A
C:\Users\GeeksForGeeks>
```

Чтобы узнать больше о полиморфизме, пожалуйста, обратитесь к [Полиморфизм в Perl](#)

Наследование: -

Наследование - это способность любого класса извлекать и использовать функции других классов. Это процесс, посредством которого новые классы, называемые производными классами, создаются из существующих классов, называемых базовыми классами. Наследование в Perl может быть реализовано с использованием [пакетов](#). Пакеты используются для создания родительского класса, который может использоваться в производных классах для наследования функциональных возможностей.

Чтобы узнать больше о наследовании, пожалуйста, обратитесь к [Наследование в Perl](#)

Инкапсуляция: -

Инкапсуляция - это процесс объединения данных для защиты их от внешних источников, которым необязательно иметь доступ к этой части кода. Технически при инкапсуляции переменные или данные класса скрыты от любого другого класса и могут быть доступны только через любую функцию-член собственного класса, в котором они объявлены. Этот процесс также называется **сокрытием данных**. Чтобы узнать больше об инкапсуляции, пожалуйста, обратитесь к [Инкапсуляции в Perl](#)

Абстракция: -

Абстракция - это процесс, посредством которого пользователь

получает доступ только к существенным деталям программы, а тривиальная часть скрыта от пользователя. Пример: Автомобиль рассматривается как машина, а не как отдельные ее компоненты. Пользователь может знать только то, что делается, но не часть того, как это делается. Это и есть абстракция.

Чтобы узнать больше об абстракции, пожалуйста, обратитесь к

[Абстракция в Perl](#)

Подпрограммы

Что такое подпрограммы?

Функция или подпрограмма Perl - это группа операторов, которые вместе выполняют определенную задачу. В каждом языке программирования пользователь хочет повторно использовать код. Таким образом, пользователь помещает раздел кода в функцию или подпрограмму, чтобы не было необходимости писать код снова и снова.

Пример:

```
#!/usr/bin/perl

# Perl Program to demonstrate the
# subroutine declaration and calling

# defining subroutine
sub ask_user
{
    print "Hello Geeks!\n";
}

# calling subroutine
# you can also use
# &ask_user();
ask_user();
```

Вывод:

```
Hello Geeks!
```

Несколько подпрограмм

Несколько подпрограмм в Perl можно создать, используя ключевое слово 'multi'. Это помогает в создании нескольких подпрограмм с

одинаковым именем.

Пример:

```
multi Func1($var){statement};  
multi Func1($var1, $var2){statement1; statement2;}
```

Пример:

```
#!/usr/bin/perl  
# Program to print factorial of a number  
  
# Factorial of 0  
multi Factorial(0)  
{  
    1; # returning 1  
}  
  
# Recursive Function  
# to calculate Factorial  
multi Factorial(Int $n where $n > 0)  
{  
    $n * Factorial($n - 1); # Recursive Call  
}  
  
# Printing the result  
# using Function Call  
print Factorial(15);
```

Output:

3628800

To know more about Multiple Subroutines, please refer to [Multiple Subroutines in Perl](#)

Модули и пакеты

Модуль в Perl представляет собой набор связанных подпрограмм и переменных, которые выполняют набор задач программирования. Модули Perl можно использовать повторно. Модуль Perl - это пакет, определенный в файле с тем же именем, что и у пакета, и имеющий расширение .pm. Пакет Perl представляет собой набор кода, который находится в своем собственном пространстве имен.

Для импорта модуля мы используем функции `require` или `use`. Для доступа к функции или переменной из модуля используется `::`.

Примеры:

```
#!/usr/bin/perl

# Using the Package 'Calculator'
use Calculator;

print "Enter two numbers to multiply";

# Defining values to the variables
$a = 5;
$b = 10;


# Subroutine call
Calculator::multiplication($a, $b);

print "\nEnter two numbers to divide";

# Defining values to the variables
$a = 45;
$b = 5;

# Subroutine call
Calculator::division($a, $b);
```

Вывод:



```
C:\Users\GeeksForGeeks>perl Test.pl
Enter two numbers to multiply
***Multiplication is 50
Enter two numbers to divide
***Division is 9
C:\Users\GeeksForGeeks>
```

Ссылки

Ссылка в Perl - это скалярный тип данных, который содержит местоположение другой переменной. Другой переменной могут быть скалярные значения, хэши, массивы, имена функций и т.д. Ссылку

можно создать, добавив к ней обратную косую черту.

Пример:

```
# Perl program to illustrate the
# Referencing and Dereferencing
# of an Array

# defining an array
@array = ('1', '2', '3');

# making an reference to an array variable
$reference_array = \@array;

# Dereferencing
# printing the value stored
# at $reference_array by prefixing
# @ as it is a array reference
print @$reference_array;
```

Вывод:

123

Чтобы узнать больше о ссылках, пожалуйста, обратитесь к [Ссылкам в Perl](#)

Регулярное выражение

Регулярное выражение (Regex или Regexp или RE) в Perl - это специальная текстовая строка для описания шаблона поиска в заданном тексте. Регулярное выражение в Perl связано с основным языком и отличается от PHP, Python и т.д.

В основном операторы привязки используются с оператором `m//`, чтобы можно было подобрать требуемый шаблон.

Пример:

```
# Perl program to demonstrate
# the m// and =~ operators

# Actual String
$a = "GEEKSFORGEEKS";

# Prints match found if
# its found in $a
if ($a =~ m[GEEKS])
```

```
{  
    print "Match Found\n";  
}  
  
# Prints match not found  
# if its not found in $a  
else  
{  
    print "Match Not Found\n";  
}
```

Вывод:

Match Found

Вывод:

Match Found

Кванторы в регулярных выражениях

Perl предоставляет несколько количественных показателей регулярных выражений, которые используются для указания того, сколько раз данный символ может быть повторен до выполнения сопоставления. В основном это используется, когда неизвестно количество символов, которые будут сопоставлены.

Существует шесть типов кванторов Perl, которые приведены ниже:

- ***** = Здесь говорится, что компонент должен присутствовать ноль или более раз.
- **+** = Здесь говорится, что компонент должен присутствовать один или несколько раз.
- **?** = Здесь говорится, что компонент должен присутствовать либо ноль, либо один раз.
- **{n}** = Здесь говорится, что компонент должен присутствовать n раз.
- **{n, }** = Здесь говорится, что компонент должен присутствовать не менее n раз.
- **{n, m}** = Здесь говорится, что компонент должен присутствовать не менее n раз и не более m раз.

Обработка файлов

В Perl дескриптор файла связывает имя с внешним файлом, который можно использовать до завершения программы или пока дескриптор файла не будет закрыт. Короче говоря, дескриптор файла подобен соединению, которое можно использовать для изменения содержимого внешнего файла, и этому соединению присваивается имя (дескриптор файла) для более быстрого доступа и удобства.

Три основных дескриптора файлов в Perl - это STDIN, STDOUT и STDERR, которые представляют устройства стандартного ввода, стандартного вывода и стандартной ошибки соответственно.

Чтение из файла и запись в файл с помощью FileHandle

Чтение из дескриптора файла можно выполнить с помощью функции печати.

```
# Opening the file
open(fh, "GFG2.txt") or die "File '$filename' can't be opened";

# Reading First line from the file
$firstline = <fh>;
print "$firstline\n";
```

Вывод :

A screenshot of a Windows Command Prompt window. The title bar says "Select Command Prompt". The command prompt shows the directory "C:\Users\GeeksForGeeks" and the command "perl File.pl" being executed. The output of the script is "GeeksforGeeks". The prompt then returns to "C:\Users\GeeksForGeeks>".

```

C:\Users\GeeksForGeeks>perl File.pl
GeeksforGeeks
C:\Users\GeeksForGeeks>
```

Запись в файл также может быть выполнена с помощью функции печати.

```
# Opening file Hello.txt in write mode
open (fh, ">", "Hello.txt");

# Getting the string to be written
```

```
# to the file from the user
print "Enter the content to be added\n";
$a = <>;

# Writing to the file
print fh $a;

# Closing the file
close(fh) or "Couldn't close the file";
```

Выполнение кода для написания:



```
Command Prompt
C:\Users\GeeksForGeeks>perl File.pl
Enter the content to be added
Welcome to GeeksForGeeks!!!
C:\Users\GeeksForGeeks>
```

Обновленный файл:



```
Hello - Notepad
File Edit Format View Help
Welcome to GeeksForGeeks!!!
```

Основные операции с файлами

С файлами с помощью файловых дескрипторов можно выполнять множество операций. Это:

- Открытие и чтение файла
- Запись в файл
- Добавление в файл

- Чтение CSV-файла

Операторы тестирования файлов

Операторы тестирования файлов в Perl - это логические операторы, которые возвращают значения True или False. В Perl есть много операторов, которые вы можете использовать для тестирования различных аспектов файла. Например, для проверки существования файла используется оператор `e`.

В следующем примере используется `'-e'`, оператор существования, чтобы проверить, существует файл или нет:

```
#!/usr/bin/perl

# Using predefined modules
use warnings;
use strict;

# Providing path of file to a variable
my $filename = 'C:\Users\GeeksForGeeks\GFG.txt';

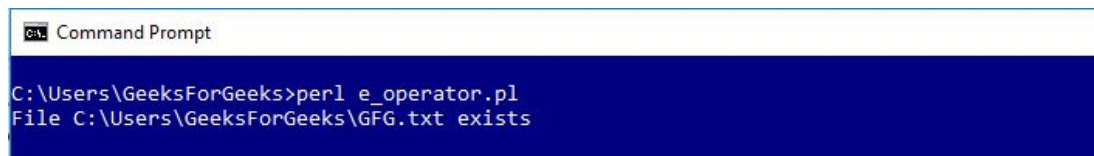
# Checking for the file existence
if(-e $filename)
{

    # If File exists
    print("File $filename exists\n");
}

else
{

    # If File doesn't exists
    print("File $filename does not exists\n");
}
```

Вывод:



```
Command Prompt
C:\Users\GeeksForGeeks>perl e_operator.pl
File C:\Users\GeeksForGeeks\GFG.txt exists
```

Чтобы узнать больше о различных операторах при тестировании файлов, пожалуйста, обратитесь к [Операторы тестирования файлов в Perl](#)

Работа с файлами Excel

Файлы Excel являются наиболее часто используемым офисным приложением для взаимодействия с компьютерами. Для создания файлов Excel с помощью Perl вы можете использовать padre IDE, мы также будем использовать модуль Excel::Writer::XLSX. Perl использует функцию write() для добавления содержимого в файл Excel.

Создание файла Excel:

Файлы Excel можно создавать с помощью командной строки Perl, но сначала нам нужно загрузить модуль Excel::Writer::XLSX.

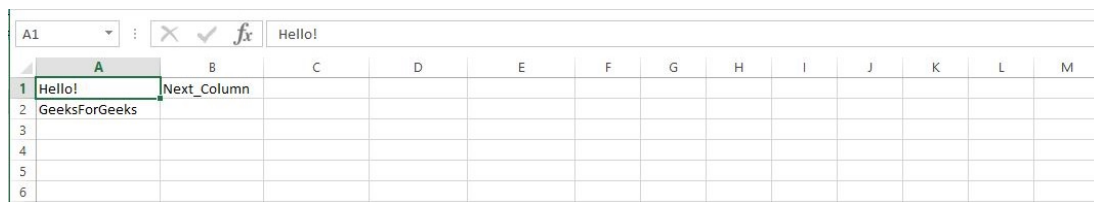
```
#!/usr/bin/perl
use Excel::Writer::XLSX;

my $Excelbook = Excel::Writer::XLSX->new( 'GFG_Sample.xlsx' );
my $Excelsheet = $Excelbook->add_worksheet();

$Excelsheet->write( "A1", "Hello!" );
$Excelsheet->write( "A2", "GeeksForGeeks" );
$Excelsheet->write( "B1", "Next_Column" );

$Excelbook->close;
```

Вывод:



The screenshot shows an Excel spreadsheet with the following data:

	A	B	C	D	E	F	G	H	I	J	K	L	M
1	Hello!	Next_Column											
2	GeeksForGeeks												
3													
4													
5													
6													

Чтение из файла Excel:

Чтение файла Excel на Perl выполняется с помощью `Spreadsheet::Read` модуля в скрипте Perl. Этот модуль экспортирует ряд функций, которые вы либо импортируете, либо используете в своем скрипте Perl code. `ReadData()` функция используется для чтения из файла Excel.

Пример:

```
use 5.016;
use Spreadsheet::Read qw(ReadData);
my $book_data = ReadData ('new_excel.xlsx');
say 'A2: ' . $book_data->[1]{A2};
```

Вывод:

A2: GeeksForGeeks

Обработка ошибок

Обработка ошибок в [Perl](#) - это процесс принятия соответствующих мер против программы, которая вызывает трудности при выполнении из-за какой-либо ошибки в коде или компиляторе. Например, при открытии файла, который не существует, возникает ошибка, или при обращении к переменной, которая не была объявлена, возникает ошибка.

Программа остановится при возникновении ошибки, и, таким образом, используя обработку ошибок, мы можем предпринять соответствующие действия вместо полной остановки программы.

Обработка ошибок является основным требованием для любого языка, склонного к ошибкам.

Perl предоставляет две встроенные функции для генерации фатальных исключений и предупреждений, которые:

1. `die()`
2. `warn()`

`die()`: сигнализировать о возникновении фатальных ошибок в том смысле, что данной программе не следует позволять продолжать работу.

Например, доступ к файлу с помощью `open()` сообщает, прошла ли операция открытия успешно, прежде чем переходить к другим файловым операциям.

```
open FILE, "filename.txt" or die "Cannot open file: $!\n";
```

`warn()`: в отличие от `die()` функции, `warn()` генерирует предупреждение вместо фатального исключения.

Например:

```
open FILE, "filename.txt" or warn "Cannot open file: $!\n";
```

Чтобы узнать больше о различных методах обработки ошибок, пожалуйста, обратитесь к [Обработка ошибок в Perl](#)

Готовитесь ли вы к своему первому собеседованию при приеме на работу или стремитесь повысить квалификацию в этом постоянно развивающемся технологическом мире, [курсы GeeksforGeeks](#) - ваш ключ к успеху. Мы предоставляем контент высочайшего качества по доступным ценам, направленный на ускорение вашего роста в сжатые сроки. Присоединяйтесь к миллионам тех, кого мы уже поддержали, и мы здесь, чтобы сделать то же самое для вас. Не пропустите - [проверьте это сейчас!](#)

Ищете место, где можно поделиться своими идеями, учиться и общаться? Наш портал [сообщества](#) - это как раз то, что вам нужно!

Приходите [присоединяйтесь к нам](#) и посмотрите, о чем весь этот ажиотаж!

Испытание "Три 90" заканчивается 5 февраля! Последний шанс получить 90% возмещения, пройдя 90% курс за 90 дней. [Изучите](#) предложение сейчас.

Последнее обновление : 27 октября 2021 г.

5

[Предыдущая страница](#)

[Далее](#)

Набор модулей Perl | DBI
(интерфейс, независимый от базы
данных) - 1

Perl | Рекурсивные подпрограммы

[Поделитесь своими мыслями в
комментариях](#)

[Добавить свой
комментарий](#)

Похожие чтения

Perl | Базовый синтаксис программы
на Perl

Функция Perl | `chomp()`

[Perl | Отслеживание возврата в
регулярном выражении](#)

Perl | Поиск в файле с помощью
регулярного выражения

Perl | Операторы | Set - 1

Оператор Perl | `lt`

Функция Perl | `chop()`

Функция Perl | `rename()`

Perl | Подпрограммы или функции

Perl | Принятие решений (`if`, `if-else`,
Вложенная лестница `if`, `if-elsif`, `если
только`, `если только-else`, `если...`

A **Abhinav96**[Подписаться](#)Теги статей : [Perl](#)

Дополнительная информация



GeeksforGeeks
Soochit Education Private Limited
A-143, 9th Floor, Sovereign
Corporate Tower, Sector-136,
Noida, Uttar Pradesh - 201305



Company

[About Us](#)[Legal](#)[Careers](#)[In Media](#)[Contact Us](#)[Advertise with us](#)[GFG Corporate Solution](#)[Placement Training Program](#)[Apply for Mentor](#)

Explore

[Job-A-Thon Hiring Challenge](#)[Hack-A-Thon](#)[GfG Weekly Contest](#)[Offline Classes \(Delhi/NCR\)](#)[DSA in JAVA/C++](#)[Master System Design](#)[Master CP](#)[GeeksforGeeks Videos](#)[Geeks Community](#)

Languages

[Python](#)
[Java](#)
[C++](#)
[PHP](#)
[GoLang](#)
[SQL](#)
[R Language](#)
[Android Tutorial](#)
[Tutorials Archive](#)

Data Science & ML

[Data Science With Python](#)
[Data Science For Beginner](#)
[Machine Learning Tutorial](#)
[ML Maths](#)
[Data Visualisation Tutorial](#)
[Pandas Tutorial](#)
[NumPy Tutorial](#)
[NLP Tutorial](#)
[Deep Learning Tutorial](#)

Python

[Python Programming Examples](#)
[Django Tutorial](#)
[Python Projects](#)
[Python Tkinter](#)
[Web Scraping](#)
[OpenCV Python Tutorial](#)
[Python Interview Question](#)

DevOps

[Git](#)
[AWS](#)
[Docker](#)
[Kubernetes](#)
[Azure](#)

DSA

[Data Structures](#)
[Algorithms](#)
[DSA for Beginners](#)
[Basic DSA Problems](#)
[DSA Roadmap](#)
[Top 100 DSA Interview Problems](#)
[DSA Roadmap by Sandeep Jain](#)
[All Cheat Sheets](#)

HTML & CSS

[HTML](#)
[CSS](#)
[Web Templates](#)
[CSS Frameworks](#)
[Bootstrap](#)
[Tailwind CSS](#)
[SASS](#)
[LESS](#)
[Web Design](#)

Computer Science

[GATE CS Notes](#)
[Operating Systems](#)
[Computer Network](#)
[Database Management System](#)
[Software Engineering](#)
[Digital Logic Design](#)
[Engineering Maths](#)

Competitive Programming

[Top DS or Algo for CP](#)
[Top 50 Tree](#)
[Top 50 Graph](#)
[Top 50 Array](#)
[Top 50 String](#)

[GCP](#)[Top 50 DP](#)[DevOps Roadmap](#)[Top 15 Websites for CP](#)

System Design

JavaScript

[High Level Design](#)[JavaScript Examples](#)[Low Level Design](#)[TypeScript](#)[UML Diagrams](#)[ReactJS](#)[Interview Guide](#)[NextJS](#)[Design Patterns](#)[AngularJS](#)[OOAD](#)[NodeJS](#)[System Design Bootcamp](#)[Lodash](#)[Interview Questions](#)[Web Browser](#)

NCERT Solutions

School Subjects

[Class 12](#)[Mathematics](#)[Class 11](#)[Physics](#)[Class 10](#)[Chemistry](#)[Class 9](#)[Biology](#)[Class 8](#)[Social Science](#)[Complete Study Material](#)[English Grammar](#)

Commerce

UPSC Study Material

[Accountancy](#)[Polity Notes](#)[Business Studies](#)[Geography Notes](#)[Economics](#)[History Notes](#)[Management](#)[Science and Technology Notes](#)[HR Management](#)[Economy Notes](#)[Finance](#)[Ethics Notes](#)[Income Tax](#)[Previous Year Papers](#)

SSC/ BANKING

Colleges

[SSC CGL Syllabus](#)[Indian Colleges Admission & Campus Experiences](#)[SBI PO Syllabus](#)[List of Central Universities - In India](#)[SBI Clerk Syllabus](#)[Colleges in Delhi University](#)[IBPS PO Syllabus](#)[IIT Colleges](#)[IBPS Clerk Syllabus](#)[NIT Colleges](#)[SSC CGL Practice Papers](#)

Companies

META Owned Companies
Alphabhet Owned Companies
TATA Group Owned Companies
Reliance Owned Companies
Fintech Companies
EdTech Companies

Exams

JEE Mains
JEE Advanced
GATE CS
NEET
UGC NET

Free Online Tools

Typing Test
Image Editor
Code Formatters
Code Converters
Currency Converter
Random Number Generator
Random Password Generator

Preparation Corner

Company Wise Preparation
Preparation for SDE
Experienced Interviews
Internship Interviews
Competitive Programming
Aptitude Preparation
Puzzles

More Tutorials

Software Development
Software Testing
Product Management
SAP
SEO - Search Engine Optimization
Linux
Excel

Write & Earn

Write an Article
Improve an Article
Pick Topics to Write
Share your Experiences
Internships

@GeeksforGeeks, Sanchhaya Education Private Limited, All rights reserved