1 Работа с числами

1.1 Нотации

Числа можно задавать в разных нотациях

Нотация	Пример
Целочисленная	10000000
	10_000_000
С разделителем	10.257
Scientific	1.5e−14
Шестнадцатеричная	0x7af5
Восьмеричная	010
Двоичная	0b10

1.2 Предостережение

Так как числа в компьютере представляются в виде послдовательности битов, при работе с числами (используя стандартные средства) следует помнить о нескольких вещах:

1. Максимальное целое число с которым может работать компьютер ограничивается длинной машинного слова (количеством битов, которыми записывается число): в 32 разрядных компьютерах это приблизительно два милиарда, в 64 разрядных компьютрах это в два милиарда раз больше.

2. Для десятичных дробей нет такого ограничения, но точность представления дроби падает с её ростом.

1.3 функции

Также как и для строк, для работы с числами есть стандартный набор функций:

функция	описание	пример	результат
abs	абсолютное значение	abs -25	25
sin	синус (в радианах)	sin 3.14	0.00159
cos	косинус (в радианах)	cos 3.14	-0.9999
exp	экспонента	exp 3	20.08
log	натуральный логарифм	log exp 3	3
int	целая часть	int 5.99	5
rand	случайное число от 0 до аргумента (оба включительно)	rand 5	1.60
sqrt	квадратный корень	sqrt 1024	32

2 Немного о контекстах

Как и в разговорных языках, то что вы говорите (пишете) может иметь разные значения в зависимости от контекста.

2.1 Местоимения

В разговоре, если вы например обсуждаете Андрея Александровича Миронова, “Андрей Александрович Миронов” произносится только один раз (устанавливаете контекст), а дальше используете местоимения “он”, “его”, итп. В Perl аналогом такого местоимения является особая переменная $_. Если вы вызываете функцию не указывая её аргумент, то аргументом предполагается переменная $_. Это удобно, когда нужно сделать много действий с одной переменной.

Листинг 1: Программа подсчета введенных букв

```
print "Enter your text:\n";
$_ = <>; #Читаем ввод пользователя
         #записываем в переменную "по умолчанию"
chomp; #отрезаем символ новой строки
print; #Печатаем
print " contains ", length, " characters\n";
```

2.2 Булевый (логический) контекст

Черезвычайно часто, нужно чтобы програма сама принимала решения вроде: читать или не читать, писать или уже хватит. Как задавать такие точки принятия решений мы поговорим в следующем разделе. А пока нам необходимо определиться что является истиной и ложью в понимании Perl. Итак выражение является ложным, если результатом его вычисления является:

- пустая строка
- 0 (число или строка)
- неопределенное значение **undef**: иногда функция может сказать <Не знаю>.

Во всех остальных случаях значение считается истинной.

3 Условные операторы

3.1 Простая запись

В конце любого выражения (там где вы ставите точку с запятой) можно использовать один из простых логических операторов:

Оператор	название
if EXPR	если EXPR верно
while EXPR	пока EXPR верно

EXPR - здесь логическое выражение. Говоря сухим строгим языком, выражение слева от if будет вычислено только если EXPR верно. while будет вычислять выражение снова и снова пока EXPR остается верным. А если просто: if это «если», а while это «пока» и используют их также как и в

английском языке.

Разберем сказанное на примере программы, которая читает от пользователя строку и выводит её “задом на перед”.

Листинг 2: Программа выводящая строку “задом на перед”

```
$_ = <>;
chomp;
my $reversed;
#отрезаем символы пока есть, что отрезать
#и дописываем их в reversed
$reversed .= chop while $_;
print $reversed, "\n";
```

Давайте подробнее разберем, что происходит в строке со словом **while**. Функция chop отрезает последний символ от строки в переменной $_, а .= добавляет откушенный символ к строке в переменной $reversed. Это действие выполнилось бы один раз, но while выполняет его снова и снова, строка в $reversed растет, а в $_ уменьшается. Наконец в $_ не остается ни одного символа (все они перекочевали в $reversed), другими словами $_ содержит пустую строку "". А такая строка в логическом контексте является ложью. Поэтому **while** больше не вычисляет выражение слева и мы переходим к строке с **print**.

Схожим образом используется **if**, с той лишь разницой что **if** проверит условие и выполнит (или не выполнит) действие один раз.

4 Логические операторы

В Perl есть отдельные наборы операций для сравнения чисел и строк, которые удобно использовать в условных операторах.

числа	строки	названия	пример	значение
==	Eq	равно	5 == 5	1
!=	Ne	не равно	5 != 5	
>	gt	больше	4 > 5	
<	lt	меньше	4 < 5	1
>=	ge	больше или равно	4 >= 5	
<=	le	меньше или равно	5 <= 5	1

Для комбинирования условий используют операторы and, or, и not. Например вы хотите напечатать ваше число, если оно меньше пяти, но больше нуля: **print** $a **if** $a $<$ 5 and $a $>$ 0;

- Логическое ИЛИ возвращает истину[1], если истинно или выражение слева или выражение справа

[1] Точнее or возвращает значение выражения слева если оно истинно или значение выражения справа. Это поведение дает приятный побочный результат: вы можете выбрать первое выражение которое окажется истинным. Например: $WORK_VALUE = $USER_INPUT or $DEFAULT_VALUE

- Логическое И возвращает возвращает истину[2], если истинно и выражение слева и выражение справа.
- Логическое НЕ возвращает истину если выражение ложно или ложь если выражение верно. Логическое НЕ не сравнивает два выражения, а возвращает противоположный результат выражения справа: **print** "take a break!\n" **if** not $BOSS_HERE [3]

Существуют также альтернативные записи, более знакомые математикам (или програмистам на других языках):

название	оператор	альтернатива
Логическое ИЛИ	or	||
Логическое И	and	&&
Логическое НЕ	not	!

5 Составная запись условных операторов

5.1 Циклы

Гораздо чаще операторы if и while применяют сразу к нескольким выражениям. Для этого используют составную запись:

Листинг 3: пример использования цикла while

```
print "Обратный отсчет:\n"
my $i = 10;
 while($i > 0){
  print $i, "\n";
  sleep 1000; #ждем тысячу милисекунд
  $i--; # уменьшаем счетчик на единицу
}
```

Здесь три строки между фигурными скобками будут выполнены 10 раз. Внутри круглых скобок записывается условие, которое проверяет while.

Использование счетчика в цикле используется так часто, что для него придуман спецальный цикл `for`. Перепишем предыдущий код с использованием цикла `for`

Листинг 4: пример использования цикла for

```
print "Обратный отсчет:\n"
for(my $i = 10; $i>0; $i--){
  print $i, "\n";
  sleep 1000 ; #ждем тысячу милисекунд
}
```

[2] Значение выражения справа если истинно выражение слева, или значение выражения слева.

[3] Как и в английском языке, вместо **if** not и **while** not можно использовать **unless** и **until**

Внутри круглых скобок записываются три выражения, разделенные «;».

- Первое выражение выполняется только один раз, при входе в цикл, его еще называют инициализацией.
- Второе выражение, как и в случае с `while`, проверяется на истинность в начале каждой новой итерации.
- Третье выражение вычисляется в конце каждой итерации.

5.2 Ветвления

Бывает, что программе необходимо принять одно из нескольких решений. Например, если число отрицательное, то делать одно, если положительное – делать второе, а если нуль – то делать третье. Конечно в этом случае можно было бы применить несколько if подряд:

Листинг 5: Нерациональное ветвление

```
$number = <>;
chomp $number;
if ($number > 0) {...}
if ($number < 0) {...}
if ($number == 0) {...}
```

Но тогда при любом введенном числе, `$number` будет сравниваться с нулем все три раза, что не рационально.

После оператора `if` в составном виде, можно указать несколько операторов `elsif` и/или оператор `else`. Эти операторы не будут проверять свое условие если предыдущий if или elsif получил истину. Перепишем предыдущий листинг:

Листинг 6: Рациональное ветвление

```
$number = <>;
chomp $number;
if ($number > 0) {...}
elsif ($number < 0) {...}
else {...}
```

При такой записи `$number` будет сравниваться с нулем максимум два раза.

6 Задачи

На один балл:

1. Написать программу которая вычисляет число пи. Делает она это так: кидает случайную точку на прямоугольник [0,1]*[0,1]. Если точка попала в круг радиусом 1 и центром в (0,0) то увеличивает соответствующую переменную на 1. Площадь круга пропорциональна величине счетчика. Программа должна спросить число итераций у пользователя.

2. Написать программу которая просит пользователя отгадать пятизначное целое число по цифрам. Программа должна сообщать удалось или не удалось пользователю отгадать текущую цифру. В конце программа выводит поздравление с победой и число.

3. Написать программу которая создает случайное число не более 1000 и просит пользователя отгадать его. Отгадывание происходит так: пользователь вводит число, если то число что он ввел больше загаданного то программа печатает '>', если меньше то '<' если пользователь угадал программа выводит поздравление.

на 2 балла:

1. Написать текстовый калькулятор. Программа должна спросить число, потом спросить действие которое надо выполнить (+,-,*,/,sin,ln,exit (выход из программы) и с сбросить текущее значение) в текстовой форме (т.е. чтобы посчитать ln надо ввести 'ln'). Далее если необходимо попросить ввести второе число, вывести результат. Далее программа должна ждать ввода действия, используя результат полученный в последнем вычислении в качестве числа к которому надо его применить. Пример общения с программой:

   first value:10
   operation:-
   seconв value:2
   result = 8
   operation:+
   second value:4
   result = 12
   operation:t
   't' is not valid operation! operation:ln
   result = 2.484906649788
   operation:c
   first value:5
   operation:+
   second value:1
   result = 6
   operation:exit