1. Strings

Introduction

```
// in pike only double quotes are used for strings

// they are not interpolated.

// single quotes are used for chars (the integer value of a character)

// see chapter 1.4

//------------------------------

string str;                     // declare a variable of type string

str = "\n";                     // a "newline" character

str = "Jon \"Maddog\" Orwant";  // literal double quotes

//------------------------------

str =
#"This is a multiline string
terminated by a double-quote like any other string";
//------------------------------
```

Accessing Substrings

```
// accessing part of a string

//------------------------------

string str, value;
int offset, count;
```

```
value = str[offset..offset+count];
value = str[offset..];

string newstring, newtail;
str = str[..offset-1]+newstring+str[offset+count..];
str = str[..offset-1]+newtail;

//------------------------------

// get a 5-byte string, skip 3, then grab 2 8-byte strings, then the rest

string leading, s1, s2, trailing;
[leading, s1, s2, trailing] = array_sscanf(str, "%5s%*3s%8s%8s%s");

// split at five byte boundaries

array(string) fivers = str/5;

// chop string into individual characters

array(string) chars = str/"";

//------------------------------


str = "This is what you have";

string first, start, rest, last, end, piece;
int t = str[0];
// 84

first = str[0..0];
// "T"

start = str[5..5+1];
// "is"

rest  = str[13..];
// "you have"

last  = str[sizeof(str)-1..sizeof(str)-1];
// "e"

end   = str[sizeof(str)-4..];
// "have"

piece = str[sizeof(str)-8..sizeof(str)-8+2];
// "you"


str = "This is what you have";
str = replace(str, ([ " is ":" wasn't " ]) );
```

```
// "This wasn't what you have"

str = str[..sizeof(str)-13]+"ondrous";
// "This wasn't wondrous"

str = str[1..];
// "his wasn't wondrous"

str = str[..sizeof(str)-11];
// "his wasn'"

str = "This is what you have";
str = replace(str[..4], ([ "is":"at" ]) )+str[5..];
// "That is what you have"

str = "make a hat";
// "make a hat"

[str[0], str[-1]] = ({ str[-1], str[0] });
// "take a ham"


string a, b, c;
a = "To be or not to be";
b = a[6..11];
// "or not"

b = a[6..7]; c=a[3..4];
write("%s\n%s\n", b, c);
/*
or
be
*/
//-------------------------------------------------


string cut2fmt(int ... positions)
{
  string template = "";
  int lastpos  = 1;
  foreach(positions ;; int place)
  {
    template += "A" + (place - lastpos) + " ";
    lastpos = place;
  }
  template += "A*";
  return template;
}

string fmt = cut2fmt(8, 14, 20, 26, 30);
write("%s\n", fmt);
```

```
//A7 A6 A6 A6 A4 A*
```

Establishing a Default Value

```
// set a default, ie, only set the value if no other value is set.

//-----------------------------

// use b if b is true, else c

a = b || c;

// set x to y unless x is already true

if(!x)
  x = y;

// use b if b is defined, else c

// an undefined variable would be a compile time error so this

// does not really apply.


// return b if b is defined (was supplied by the caller), else c

int foo(int c, int|void b)
{
  return zero_type(b) ? c : b;
}

foo = bar || "DEFAULT VALUE";
argv = argv[1..];               // remove program, as that is always set.

dir = argv[0] || "/tmp";        // and see if anything is left...

dir = sizeof(argv) ? argv[0] : "/tmp";
count[shell||"/bin/sh"]++;

user = getenv("USER") || getenv("LOGNAME") || getpwuid(getuid())[0] ||
"Unknown uid number "+getuid();

if(!starting_point)
  starting_point = "Greenwich";

if(!sizeof(a))
  a = b;              // copy only if empty
```

```
a = (sizeof(b)?b:c);  // assign b if nonempty, else c
```

Exchanging Values Without Using Temporary Variables

```
[var1, var2] = ({ var2, var1 });  // gee, i love this example.
                                  // it didn't even occur to me before
                                  // :-)

temp = a;
a    = b;
b    = temp;

a = "alpha";
b = "omega";
[a, b] = ({ b, a });

[alpha, beta, production] = "January March August"/" ";
[alpha, beta, production] = ({ beta, production, alpha });
```

Converting Between ASCII Characters and Values

```
// print the ascii value of a char, or the char from its ascii value

int i;                          // declare a variable of type int

i = 'a';                        // the ascii value of "a"

i = '\n';                       // the ascii value of a "newline"

//-----------------------------

string char = "foo";
int num = char[0];   // gets the ascii value from the first char (that's
                     // what ord() in perl does)

char = String.int2char(num);

char = sprintf("%c",num);   // the same as String.int2char(num) :-)
```

```
write("Number %d is character %[0]c\n", num);

Number 101 is character e

string str;
array(int) arr;
arr = (array)str;
str = (string)arr;
int ascii_value = 'e';                          // now 101

string character = String.int2char(101);    // now "e"


write("Number %d is character %[0]c\n", 101);


array(int) ascii_character_numbers = (array(int))"sample";
write("%s\n", (array(string))ascii_character_numbers*" ");

string word = (string)ascii_character_numbers;
string word = (string)({ 115, 97, 109, 112, 108, 101 });  // same

write(word+"\n");
// sample


string hal ="HAL";
array(int) ascii = (array)hal;
array(int) ibm = ascii[*]+1;        // add 1 to each element in the array.

array(int) ibm = map(ascii, `+, 1) // apply the function +, with the argum

                                    // 1, to each element in the array.

write(ibm+"\n");                    // prints "IBM"
```

Processing a String One Character at a Time

```
string hello = "Hello world!";
array(string) chars = hello/"";     // array of characters as strings


foreach(chars;; string char)        // this also matches newlines

  ;  // do stuff with char

//-----------------------------
```

```
string data = "an apple a day";
array(string) chars = data/"";
mapping(string:int) seen = ([]);

foreach(chars ;; string char)
  seen[char]++;

write("unique chars are: %s\n", sort(indices(seen))*"");
// unique chars are:  adelnpy

//------------------------------

string data = "an apple a day";
string result = sort(indices(mkmapping(data/"", allocate(sizeof(data))))*"

write("unique chars are: %s\n", result);
// unique chars are:  adelnpy

//------------------------------

string data = "an apple a day";
int sum;

foreach(data ;; int char)
  sum += char;

write("sum is %d\n", sum);
// sum is 1248

//------------------------------

string data = "an apple a day";
int sum=`+(@(array)data);

write("sum is %d\n", sum);
// sum is 1248

//------------------------------

// download the following standalone program
#/usr/bin/pike
// chapter 1.5
void main(int argc, array(string) argv)
{
  string data = Stdio.read_file(argv[1]);
  int checksum;

  foreach(data ;; int char)
    checksum += char;

  checksum %= pow(2,16)-1;
  write("%d\n", checksum);
```

```
}

//-----------------------------

// alternate version

// download the following standalone program
#!/usr/bin/pike
// chapter 1.5
void main(int argc, array(string) argv)
{
  string data=Stdio.read_file(argv[1]);
  int checksum = `+(@(array)data) % ((1<<16)-1);
  write("%d\n", checksum);
}

//-----------------------------

// download the following standalone program
#!/usr/bin/pike
// chapter 1.5
// slowcat - emulate a   s l o w  line printer
// usage: slowcat [-DELAY] [files ...]
void main(int argc, array argv)
{
  array(string) files;
  int delay = 1;

  if(argv[1][0] == '-')
  {
    files = argv[2..];
    delay = (int)argv[1][1..];
  }
  else
    files = argv[1..];

  foreach(files, string file)
  {
    string data = Stdio.read_file(file);
    foreach(data/"", string char)
    {
      write(char);
      sleep(0.005*delay);
    }
  }
}
```

Reversing a String by Word or Character

```
// #1.6 (reverse a string by char/word)

// by Olivier Girondel


string s = "This is  a string";
// Result: "This is  a string"


reverse(s);
// Result: "gnirts a  si sihT"


reverse(s/" ") * " ";                // preserve whitespace

// Result: "string a  is This"


(reverse(s/" ")-({ "" })) * " ";     // collapse whitespace

// Result: "string a is This"

//------------------------------

string word = "reviver";
int is_palindrome = word==reverse(word);
//------------------------------

// download the following standalone program
#!/usr/bin/pike
// chapter 1.6
void main(int argc, array(string) argv)
{
  string data=Stdio.read_file(argv[1]);
  foreach(data/"\n", string line)
  {
    if(line==reverse(line) && sizeof(line)>5)
    write("%s\n", line);
  }
}
```

Expanding and Compressing Tabs

```
string s = "This         is                a    \n   string";
```

```
string notabs=String.expand_tabs(s);
// Result: "This     is                a    \n   string"


string notabs=String.expand_tabs(s, 4);
// Result: "This     is            a    \n   string"


string notabs=String.expand_tabs(s, 4, "-");
// Result: "This   - is --------   a ---\n   string"

//----------------------------

string s = "This     is      a     string";
string tabs="";

foreach(s/8.0 ;; string stop)
{
  int spaces=sizeof(String.common_prefix(({ reverse(stop), "        "})));
  tabs+=stop[..7-spaces];
  if(spaces)
    tabs+="^I";
}
// Result: "This\t is\t a     string"


string notabs="";
foreach(tabs/"^I" ;; string stop)
{
  notabs+=stop;
  if(sizeof(stop)<8)
    notabs+=" "*(8-sizeof(stop));
}
// Result: "This     is      a     string"
```

Expanding Variables in User Input

```
// since variable names in pike do not have a special notation we need to

// "invent" one for this.

// there are a few ways to solve this problem.

// here is one:


mapping(string:string) vars = ([ "$fruit$":"apple", "$desert$":"pudding" ]
string template  = "Todays fruit is $fruit$, and for desert we have $deser
```

```
string menu = replace(template, vars);

// Result: "Todays fruit is apple, and for desert we have pudding"
```

Controlling Case

```
string upper, lower, result;
upper = "DON'T SHOUT!";
result = lower_case(upper);
// Result: "don't shout!"

//------------------------------

lower = "speak up";
result = upper_case(lower);
// Result: "SPEAK UP"

//------------------------------

result = String.capitalize(lower);
// Result: "Speak up"


//------------------------------

string text = "thIS is a loNG liNE";
array(string) words = text/" ";       // splits the line into words

words = lower_case(words[*]);         // lower_case each word

words = String.capitalize(words[*]);  // capitalize each word

text = words*" ";                     // join back

// you may do the same in one short line:

text = String.capitalize(lower_case((text/" ")[*])[*])*" ";

// download the following standalone program
#!/usr/bin/pike
// chapter 1.9
// randcap: filter to randomly capitalize 20% of the letters

void main()
{
  string input;
  while(input=Stdio.stdin.read(1))
```

```
    write(randcap(input));
}

string randcap(string char)
{
  if(random(100)<20)
    char=String.capitalize(char);
  return char;
}
```

Interpolating Functions and Expressions Within Strings

```
// since pike does not provide any string interpolation

// there are no sneaky tricks here.

// a solution could be similar to the one in chapter 1.8

// putting functions into the mapping instead of string values, or use xml

// callbacks

// TODO: provide an example of using the xml parser here
```

Indenting Here Documents

```
// we believe that indenting the string and then removing that indent does

// actually enhance readability of the code.

// but if you insist the following will remove all whitespace at the begin

// of each line:

string here=#"your text
              goes here";

string there=array_sscanf((here/"\n")[*], "%*[\t ]%s")[*][0]*"\n";

// expanded version:

array tmp=({});
```

```
foreach(here/"\n";; string line)
{
  tmp+=array_sscanf(line, "%*[\t ]%s");
}
string there=tmp*"\n";
```

Reformatting Paragraphs

```
// pike sprintf() provides a facility for wrapping (column mode):

// sprintf("%-=<int width>s", text);

// download the following standalone program
#!/usr/bin/pike
// chapter 1.12
// wrapdemo - show how wrapping with sprintf works
void main()
{
  array(string) input = ({ "Folding and splicing is the work of an editor,
                           "not a mere collection of silicon",
                           "and",
                           "mobile electrons!"});
  int columns = 20;

  write("0123456789"*2+"\n");
  write(wrap(input*" ", 20, "  ", "  ")+"\n");
}

// unlike the perl version here leadtab is relative to nexttab,
// to get a shorter lead use a negative int value. this allows the default
// to be a lead indent that is the same as nexttab, and it also has the
// advantage of allowing you to change the indent without having to worry
// the lead getting messed up.
// a negative lead will cut away from the nexttab which will be visible if
// use something other than spaces
string wrap(string text, void|int width,
            void|string|int nexttab, void|string|int leadtab)
{
  string leadindent="";
  string indent="";
  string indent2="";

  if(!width)
    width=Stdio.stdout->tcgetattr()->columns;

  if(stringp(nexttab))
  {
    indent=nexttab;
    width-=sizeof(nexttab);  // this will be off if there are chars that h
```

```
                                // different width than 1.
  }
  else if(intp(nexttab))
  {
    indent=" "*nexttab;
    width-=nexttab;
  }

  if(stringp(leadtab))
    leadindent=leadtab;
  else if(intp(leadtab))
    if(leadtab > 0)
      leadindent=" "*leadtab;
    else if(leadtab < 0)
    {
      write(indent+".\n");
      indent=indent[..(sizeof(indent)+leadtab)-1];
      write(indent+".\n");
      indent2=text[..-leadtab-1];
      text=text[-leadtab..];
    }
  return sprintf("%^s%=*s%-=*s", indent, sizeof(indent2), indent2,
                                 width, leadindent+text);
}

//-------------------------------------------------

$ ./wrapdemo
01234567890123456789
    Folding and
  splicing is the
  work of an editor,
  not a mere
  collection of
  silicon and mobile
  electrons!

// merge multiple lines into one, then wrap one long line

inherit "wrapdemo.pike";
wrap(replace(text, "\n", " "));

// read stdin and split by paragraph,

// remove \n in paragraphs

// reformat

// add paragraph break

foreach(Stdio.stdin->read()/"\n\n";; string para)
  write(wrap(replace(para, "\n", " "))+"\n\n");
```

Escaping Characters

```
// we need to escape the \ for this example, ironic, eh?

array(string) charlist=({ "%", "\\" });
string var="some input % text with \\";

// backslash

var=replace(var, charlist, "\\"+charlist[*]);

// double

var=replace(var, charlist, charlist[*]+charlist[*]);
```

Trimming Blanks from the Ends of a String

```
string line=" foo\n\t ";
array(string) many=({ " bar\n\t ", " baz\t " });

// remove spaces and tabs

line=String.trim_whites(line);
many=String.trim_whites(many[*]);

//remove spaces, tabs, newlines and carriage returns

line=String.trim_all_whites(line);
many=String.trim_all_whites(many[*]);
```

Parsing Comma-Separated Data

Soundex Matching

```
// contributed by martin nilsson


write("Lookup user: ");
string user = String.soundex(Stdio.stdin.gets());
foreach(get_all_users(), array u)
{
```

```
  string firstname="", lastname="";
  sscanf(u[4], "%s %s,", firstname, lastname);
  if( user==String.soundex(u[0]) ||
      user==String.soundex(firstname) ||
      user==String.soundex(lastname) )
    write("%s: %s %s\n", u[0], firstname, lastname);
}
```

Program: fixstyle

Program: psgrep

---

numbers.html

---

2. Numbers

Checking Whether a String Is a Valid Number

```
string number="123.3asdf";

int|float realnumber= (int)number;  // casting to int will throw away all

                                    // nonnumber parts

string rest;
[realnumber, rest] = array_sscanf(number, "%d%s"); // scan for an integer

// if rest contains anything but the empty string, then there was more tha

// number in the string
```

```
// use %f to scan for float, %x for hex or %o for octal
```

Comparing Floating-Point Numbers

```
int same(float one, float two, int accuracy)
{
  return sprintf("%.*f", accuracy, one) == sprintf("%.*f", accuracy, two);
}

int wage=536;
int week=40*wage;
write("one week's wage is: $%.2f\n", week/100.0);
```

Rounding Floating-Point Numbers

```
float unrounded=3.5;
string rounded=sprintf("%.*f", accuracy, unrounded);

float a=0.255;
string b=sprintf("%.2f", a);

write("Unrounded: %f\nRounded: %s\n", a, b);
write("Unrounded: %f\nRounded: %.2f\n", a, a);

// dec to bin

string bin=sprintf("%b", 5);

int dec=array_sscanf("0000011111111111111", "%b")[0];
                // array_sscanf returns an array


int num = array_sscanf("0110110", "%b")[0];  // num is 54

string binstr = sprintf("%b", 54);           // binstr is 110110
```

Converting Between Binary and Decimal

```
// contributed by martin nilsson.


string dec2bin(int n)
{
  return sprintf("%b",n);
}

int bin2dec(string n)
{
  return array_sscanf(n, "%b")[0];
}
```

Operating on a Series of Integers

```
// foreach(enumerate(int count, int step, int start);; int val)

// {

//   // val is set to each of count integers starting at start

// }


foreach(enumerate(y-x+1,1,x);; int val)
{
  // val is set to every integer from X to Y, inclusive

}

for(int i=x; i<=y; i++)
{
  // val is set to every integer from X to Y, inclusive

}

for(int i=x; i<=y; i+=7)
{
  // val is set to every integer from X to Y, stepsize = 7

}

foreach(enumerate(y-x+1,7,x);; int val)
{
  // val is set to every integer from X to Y, stepsize = 7
```

```
}

//----------------------------------------

write("Infancy is: ");
foreach(enumerate(3);; int val)
{
  write("%d ", val);
}
write("\n");

write("Toddling is: %{%d %}\n", enumerate(2,1,3));

write("Childhood is: ");
for (int i = 5; i <= 12; i++)
{
  write("%d ", i);
}
write("\n");

// Infancy is: 0 1 2

// Toddling is: 3 4

// Childhood is: 5 6 7 8 9 10 11 12
```

Working with Roman Numerals

```
int arabic;
string roman = String.int2roman(arabic);        // handles values up to 10


array nums=enumerate(10001);
array romans=String.int2roman(nums[*]);
mapping roman2int = mkmapping(romans, nums);

int arabic = roman2int[roman];

//-----------------------------------------------

string roman_fifteen = String.int2roman(15);    //  "XV"

write("Roman for fifteen is %s\n", roman_fifteen);

int arabic_fifteen = roman2int[roman_fifteen];
write("Converted back, %s is %d\n", roman_fifteen, arabic_fifteen);
```

```
// Roman for fifteen is XV

// Converted back, XV is 15
```

Generating Random Numbers

```
int y,x;
int rand = random(y-x+1)+x;

float y,x;
float rand = random(y-x+1)+x;

int rand = random(51)+25;
write("%d\n", rand);

array arr;
mixed elt = arr[random(sizeof(arr))];
mixed elt = random(arr);

array chars="ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz!@$%^&*"/
string password = "";
for(int i=1; i<=8; i++)
{
  password+=random(chars);
}

string password = random((({chars})*8)[*])*"";

string password = random_string(8);      // creates an untypable string


// turn the string into something typable using the base64 charset

string password = MIME.encode_base64(random_string(8))[..7];
```

Generating Different Random Numbers

```
random_seed(int seed);
random_seed((int)argv[1]);
```

Making Numbers Even More Random

```
// Crypto.Random.random(int max)

// Crypto.Random.random_string(int length)

// Crypto.Random.blocking_random_string(int length)

// Crypto.Random.add_entropy(string random_data, int entropy)
```

Generating Biased Random Numbers

```
float gaussian_rand()
{
  float u1, u2, w, g1, g2;

  do
  {
    u1 = 2.0 * random(1.0) - 1.0; u2 = 2.0 * random(1.0) - 1.0;
    w = u1 * u1 + u2 * u2;
  } while (w > 1.0);

  w = sqrt((-2.0 * log(w)) / w); g2 = u1 * w; g1 = u2 * w;

  return g1;
}

// ----


float mean = 25.0, sdev = 2.0;
float salary = gaussian_rand() * mean + sdev;

write("You have been hired at: %.2f\n", salary);
```

Doing Trigonometry in Degrees, not Radians

```
float deg2rad(float deg)
{
  return (deg / 180.0) * Math.pi;
}

float rad2deg(float rad)
```

```
{
  return (rad / Math.pi) * 180.0;
}

// ----


write("%f\n", Math.convert_angle(180, "deg", "rad"));
write("%f\n", deg2rad(180.0));

// ----------------------------


float degree_sin(float deg)
{
  return sin(deg2rad(deg));
}

// ----


float rad = deg2rad(380.0);
write("%f\n", sin(rad));
write("%f\n", degree_sin(380.0));
```

Calculating More Trigonometric Functions

```
float my_tan(float theta)
{
  return sin(theta) / cos(theta);
}

// ----


float theta = 3.7;

write("%f\n", my_tan(theta));
write("%f\n", tan(theta));
```

Taking Logarithms

```
float value = 100.0;

float log_e = log(value);
float log_10 = Math.log10(value);
```

```
// ----------------------------

float log_base(float base, float value)
{
  return log(value) / log(base);
}

// ----

float answer = log_base(10.0, 10000.0);

write("log(10, 10,000) = %f\n", answer);
```

Multiplying Matrices

```
// Pike offers a solid matrix implementation; highlights:

// * Operator overloading makes matrix operations succinct

// * Matrices may be of various types, thus allowing user to

//   choose between range representation and speed

// * Wide variety of operations available

Math.Matrix a = Math.Matrix( ({ ({3, 2, 3}), ({5, 9, 8})  }) ),
            b = Math.Matrix( ({ ({4, 7}), ({9, 3}), ({8, 1}) }) );

Math.Matrix c = a * b;

// ------------

Math.Matrix t = c->transpose();
```

Using Complex Numbers

```
// @@INCOMPLETE@@
```

Converting Between Octal and Hexadecimal

```
// Like C, Pike supports decimal-alternate notations. Thus, for example,

// the integer value, 867, is expressable in literal form as:

//

//   Hexadecimal -> 0x363

//   Octal       -> 01543

//

// For effecting such conversions using strings there is 'sprintf' and

// 'sscanf'.


int dec = 867;
string hex = sprintf("%x", dec);
string oct = sprintf("%o", dec);

// ------------


int dec;
string hex = "363"; sscanf(hex, "%x", dec);

// ------------


int dec;
string oct = "1543"; sscanf(oct, "%o", dec);

// ----------------------------


int number;

write("Gimme a number in decimal, octal, or hex: ");
sscanf(Stdio.stdin->gets(), "%D", number);

write("%d %x %o\n", number, number, number);
```

Putting Commas in Numbers

```
string commify_series(int series)
{
  return reverse((reverse((string)series) / 3.0) * ",");
}

// ----


int hits = 3452347;

write("Your website received %s accesses last month.\n", commify_series(hi

// ----------------------------


string commify(string s)
{
  function t = lambda(string m) { return reverse((reverse(m) / 3.0) * ","
  return Regexp.PCRE("([0-9]+)")->replace(s, t);
}

// ----


int hits = 3452347;
string output = sprintf("Your website received %d accesses last month.", h

write("%s\n", commify(output));
```

Printing Correct Plurals

```
string pluralise(int value, string root, void|string singular_, void|strin
{
  string singular = singular_ ? singular_ : "";
  string plural = plural_ ? plural_ : "s";

  return root + ( (value > 1) ? plural : singular );
}

// ----


int duration = 1;
write("It took %d %s\n", duration, pluralise(duration, "hour"));
write("%d %s %s enough.\n", duration, pluralise(duration, "hour"),
```

```
      pluralise(duration, "", "is", "are"));

duration = 5;
write("It took %d %s\n", duration, pluralise(duration, "hour"));
write("%d %s %s enough.\n", duration, pluralise(duration, "hour"),
      pluralise(duration, "", "is", "are"));

// ----------------------------


// Non-regexp implementation, uses the string-based, 'has_prefix'

// and 'replace' library functions

string plural(string singular)
{
  mapping(string : string) e2 =
    (["ss":"sses", "ph":"phes", "sh":"shes", "ch":"ches",
      "ey":"eys", "ix":"ices", "ff":"ffs"]);

  mapping(string : string) e1 =
    (["z":"zes", "f":"ves", "y":"ies", "s":"ses", "x":"xes"]);

  foreach(({e2, e1}), mapping(string : string) endings)
  {
    foreach(indices(endings), string ending)
    {
      if (has_suffix(singular, ending))
      {
        return replace(singular, ending, endings[ending]);
      }
    }
  }

  return singular;
}

// ----


int main()
{
  foreach(aggregate("mess", "index", "leaf", "puppy"), string word)
    write("%6s -> %s\n", word, plural(word));
}
```

Program: Calculating Prime Factors

```
// download the following standalone program
#!/usr/bin/pike
// contributed by martin nilsson

void main(int n, array args)
{
  foreach(args[1..], string arg)
  {
    mapping r = ([]);
    foreach(Math.factor((int)arg), int f)
      r[f]++;
    write("%-10s", arg);
    if(sizeof(r)==1)
      write(" PRIME");
    else
    {
      foreach(sort(indices(r)), int f)
      {
        write(" %d", f);
        if(r[f]>1) write("**%d", r[f]);
      }
    }
    write("\n");
  }
}
```

Prev Home Next
Strings Dates and Times

---

datesandtimes.html

PLEAC-Pike

Prev Next

---

3. Dates and Times

Introduction

```
// Pike has an extensive Calendar module that provides all manners of

// manipulating dates and times.

write("Today is day %d of the current year.\n", localtime(time())->yday+1)
// Today is day 325 of the current year.


write("Today is day %d of the current year.\n", Calendar.now()->year_day()
// Today is day 325 of the current year.
```

Finding Today's Date

```
int day, month, year;
mapping now=localtime(time());
year  = now->year+1900;
month = now->mon+1;
day   = now->mday;

write("The current date is %04d %02d %02d\n", year, month, day);

object now=Calendar.now();
year  = now->year_no();
month = now->month_no();
day   = now->month_day();

write("The current date is %04d %02d %02d\n", year, month, day);

write("The current date is %04d %02d %02d\n", @lambda(){ return ({ now->ye
// this is essentially the same as the respective perl code:

// lambda creates an anonymous function, which in this case takes one argu

// and returns an array. the array is the spliced into the arguments of wr

// if the goal is to get by without a temporary variable this is a rather

// pointless exercise, as there is still a temporary variable (in the func

// and the temporary function on top of that. more interresting is the
```

```
// functional approach aspect.
```

Converting DMYHMS to Epoch Seconds

```
// dwim_time() handles most common date and time formats.

Calendar.dwim_time("2:40:25 23.11.2004");
// Result: Second(Tue 23 Nov 2004 2:40:25 CET)

Calendar.dwim_time("2:40:25 23.11.2004")->unix_time();
// Result: 1101174025


Calendar.dwim_time("2:40:25 UTC 23.11.2004");
// Result: Second(Tue 23 Nov 2004 2:40:25 UTC)


// faster, because there is no need for guessing:

Calendar.parse("%Y-%M-%D %h:%m:%s %z","2004-11-23 2:40:25 UTC");
// Result: Second(Tue 23 Nov 2004 2:40:25 UTC)


// without parsing

Calendar.Second(2004, 11, 23, 2, 40, 25);
// Result: Second(Tue 23 Nov 2004 2:40:25 CET)


// functional

Calendar.Year(2004)->month(11)->day(23)->hour(2)->minute(40)->second(25);
// Result: Second(Tue 23 Nov 2004 2:40:25 CET)


Calendar.Day(2004, 11, 23)->set_timezone("UTC")->hour(2)->minute(40)->seco
// Result: Second(Tue 23 Nov 2004 2:40:25 UTC)


// set a time today

Calendar.dwim_time("2:40:25");
// Result: Second(Tue 23 Nov 2004 2:40:25 CET)

Calendar.dwim_time("2:40:25 UTC");
// Result: Second(Tue 23 Nov 2004 2:40:25 UTC)
```

```
Calendar.parse("%h:%m:%s %z","2:40:25 UTC");
// Result: Second(Tue 23 Nov 2004 2:40:25 UTC)

Calendar.Day()->set_timezone("UTC")->hour(2)->minute(40)->second(25);
// Result: Second(Tue 23 Nov 2004 2:40:25 UTC)
```

Converting Epoch Seconds to DMYHMS

```
int unixtime=1101174025;
int day, month, year;
mapping then=localtime(unixtime);
year  = then->year+1900;
month = then->mon+1;
day   = then->mday;

write("Dateline: %02d:%02d:%02d-%04d/%02d/%02d\n", then->hour, then->min,
// Dateline: 02:40:25-2004/11/23


object othen=Calendar.Second(unixtime);
// Result: Second(Tue 23 Nov 2004 2:40:25 CET)


write("Dateline: %02d:%02d:%02d-%04d/%02d/%02d\n", othen->hour_no(),
      othen->minute_no(), othen->second_no(), othen->year_no(),
      othen->month_no(), othen->month_day());
// Dateline: 02:40:25-2004/11/23
```

Adding to or Subtracting from a Date

```
int days_offet=55;
int hour_offset=2;
int minute_offset=17;
int second_offset=5;

object then=Calendar.parse("%D/%M/%Y, %h:%m:%s %p","18/Jan/1973, 3:45:50 p
            +Calendar.Day()*days_offet
            +Calendar.Hour()*hour_offset
            +Calendar.Minute()*minute_offset
            +Calendar.Second()*second_offset;
write("Then is %s\n", then->format_ctime());
// Then is Wed Mar 14 18:02:55 1973
```

```
write("To be precise: %d:%d:%d, %d/%d/%d\n",
              then->hour_no(), then->minute_no(), then->second_no(),
              then->month_no(), then->month_day(), then->year_no());
// To be precise: 18:2:55, 3/14/1973


int years   = 1973;
int months  = 1;
int days    = 18;
int offset  = 55;
object then = Calendar.Day(years, months, days)+offset;
write("Nat was 55 days old on: %d/%d/%d\n", then->month_no(), then->month_
// Nat was 55 days old on: 3/14/1973
```

Difference of Two Dates

```
int bree = 361535725;          // 16 Jun 1981, 4:35:25

int nat  = 96201950;           // 18 Jan 1973, 3:45:50


int difference = bree-nat;
write("There were %d seconds between Nat and Bree\n", difference);
// There were 265333775 seconds between Nat and Bree


int seconds =  difference                % 60;
int minutes = (difference / 60)          % 60;
int hours   = (difference / (60*60) )    % 24;
int days    = (difference / (60*60*24) ) % 7;
int weeks   =  difference / (60*60*24*7);

write("(%d weeks, %d days, %d:%d:%d)\n", weeks, days, hours, minutes, seco
// (438 weeks, 4 days, 23:49:35)


object bree = Calendar.dwim_time("16 Jun 1981, 4:35:25");
// Result: Second(Tue 16 Jun 1981 4:35:25 CEST)

object nat  = Calendar.dwim_time("18 Jan 1973, 3:45:50");
// Result: Second(Thu 18 Jan 1973 3:45:50 CET)

object difference = nat->range(bree);
// Result: Second(Thu 18 Jan 1973 3:45:50 CET - Tue 16 Jun 1981 4:35:26 CE


write("There were %d days between Nat and Bree\n", difference/Calendar.Day
// There were 3071 days between Nat and Bree
```

```
int days=difference/Calendar.Day();
object left=difference->add(days,Calendar.Day)->range(difference->end());

// Calendar handles timezone differences, and since the range crosses from

// normal to daylight savings time, there is one day which has only 23 hou

// by adding the number of days we effectively move the beginning of the r

// to the same day as the end, leaving us with a range that is less than a

// long. when adding the days, the daylight savings switch will be taken i

// account.


write("Bree came %d days, %d:%d:%d after Nat\n",
                    days,
                    (left/Calendar.Hour())%24,
                    (left/Calendar.Minute())%60,
                    (left/Calendar.Second())%60,
                    );

// Bree came 3071 days, 0:49:36 after Nat


// the following is more accurate, taking into account that the days where

// daylight savings time is switched do not have 24, but 23 and 25 hours.

// thanks to mirar on the pike list for pointing this out and providing a

// correct solution.


array(int) breakdown_elapsed(object u, void|array on)
{
  array res=({});
  if (!on) on=({Day,Hour,Minute,Second});
  foreach (on;;program|TimeRange p)
  {
    if (u==u->end()) { res+=({0}); continue; }
    int x=u/p;
    u=u->add(x,p)->range(u->end());
    res+=({x});
  }
  return res;
}

write("Bree came %d days, %d:%d:%d after Nat\n",
```

```
      @breakdown_elapsed(difference));

// Bree came 3071 days, 0:49:36 after Nat
```

Day in a Week/Month/Year or Week Number

```
mapping day=localtime(time());
day->mday;
// Result: 2

day->wday;
// Result: 4

day->yday;
// Result: 336


int year=1981;
int month=6;
int day=16;
object date;
date = Calendar.Day(year, month, day);
// Result: Day(Tue 16 Jun 1981)


date->week_day();
// Result: 3

date->week_no();
// Result: 24

date->year_day();
// Result: 167

write("%d/%d/%d was a %s\n", month, day, year, date->week_day_name());
// 6/16/1981 was a Tuesday


write("in the week number %d.\n", date->week_no());
// in the week number 25.
```

Parsing Dates and Times from Strings

```
string date = "1998-06-03";
int yyyy;
int mm;
int dd;
[yyyy, mm, dd] = array_sscanf(date, "%d-%d-%d");

object day;
day=Calendar.dwim_day(date);
day=Calendar.parse("%Y-%M-%D", date);

day->unix_time();
// Result: 896824800

day->year_no();
// Result: 1998

day->month_no();
// Result: 6

day->month_day();
// Result: 3
```

Printing a Date

```
object now=Calendar.dwim_time("Sun Sep 21 15:33:36 1997");
// Result: Second(Sun 21 Sep 1997 15:33:36 CEST)


now->format_ctime();
// Result: "Sun Sep 21 15:33:36 1997\n"


// there is no equivalent to scalar localtime


now = Calendar.Second(1973, 1, 18, 3, 45, 50);
write("strftime gives: %s %02d/%02d/%!2d\n", now->week_day_name(),
        now->month_no(), now->month_day(), now->year_no());
// strftime gives: Sunday 01/18/73

// pike doesn't have strftime, but hey.


// instead Calendar provides a large array of predefined formats:
```

```php
// format_nice() and format_nicez() depend on the unit:
now->format_nice();           // "18 Jan 1973 3:45:50"
now->week()->format_nice();   // "w3 1973"
now->format_nicez();          // "18 Jan 1973 3:45:50 CET"
now->hour()->format_nicez();  // "18 Jan 1973 3:00 CET"

// others are unit independant.
now->format_ext_time();       // "Thursday, 18 January 1973 03:45:50"
now->format_ext_ymd();        // "Thursday, 18 January 1973"
now->format_iso_time();       // "1973-01-18 (Jan) -W03-4 (Thu) 03:45:50 U
now->format_iso_ymd();        // "1973-01-18 (Jan) -W03-4 (Thu)"
now->format_mod();            // "03:45"
now->format_month();          // "1973-01"
now->format_month_short();    // "197301"
now->format_mtime();          // "1973-01-18 03:45"
now->format_time();           // "1973-01-18 03:45:50"
now->format_time_short();     // "19730118 03:45:50"
now->format_time_xshort();    // "730118 03:45:50"
now->format_tod();            // "03:45:50"
now->format_tod_short();      // "034550"
now->format_todz();           // "03:45:50 CET"
now->format_todz_iso();       // "03:45:50 UTC+1"
now->format_week();           // "1973-w3"
now->format_week_short();     // "1973w3"
now->format_iso_week();       // "1973-W03"
now->format_iso_week_short(); // "197303"
```

```
now->format_xtime();          // "1973-01-18 03:45:50.000000"

now->format_xtod();           // "03:45:50.000000"

now->format_ymd();            // "1973-01-18"

now->format_ymd_short();      // "19730118"

now->format_ymd_xshort();     // "730118"


now->format_ctime();          // "Thu Jan 18 03:45:50 1973\n"

now->format_smtp();           // "Thu, 18 Jan 1973 3:45:50 +0100"

now->format_http();           // "Thu, 18 Jan 1973 02:45:50 GMT"
```

High-Resolution Timers

```
int t=time();                 // current time in unixtime seconds

float t0=time(t);             // higher precision time passed since t

float t1=time(t);
float elapsed=t1-t0;
// Result: 0.009453


//-----------------------------------------

write("Press return when ready: ");
array(int) before=System.gettimeofday();
Stdio.stdin->gets();
array(int) after=System.gettimeofday();
int elapsed_sec=after[0]-before[0];
int elapsed_usec=after[1]-before[1];
if(elapsed_usec<0)
{
  elapsed_sec--;
  elapsed_usec+=1000000;
}

write("You took %d.%d seconds.\n", elapsed_sec, elapsed_usec);
//-------------------------------------

// this is an expanded example compared to the one given for perl
```

```
// to allow comparison of different types.

// bignum are objects of the gmp library, which are seamlessly integrated

// regular integers.


int main()
{
  // size values are adjusted so that each run takes about the same length

  gaugethis(5000000, 100, lambda(){ return random(pow(2,31)-1); });
              // values to fit into a signed 32bit int.

  gaugethis(50000, 100, lambda(){ return pow(2,64)+random(pow(2,64)); });
              // make sure values are bignum even in case 64bit ints are

  gaugethis(500000, 100, lambda(){ return random_string(10); });
              // might be interresting to compare longer strings too.

}

void gaugethis(int size, int number_of_times, function rand)
{
  array gauged_times = ({});
  float average;

  int swidth=sizeof((string)size);
  int nwidth=sizeof((string)number_of_times);
  for(int i; i<number_of_times; i++)
  {
    write("%*d: ", nwidth, i);
    array(int) arr=({});
    write("creating array: ");
    for(int j; j<size; j++)
    {
      arr += ({ rand() });
    }
    write(" sorting: ");

    float gaugetime=gauge // gauge measures cpu time, giving better result

    {
      arr=sort(arr);
    };
    gauged_times += ({ gaugetime });
    write(" %f           \r", gaugetime);
  }
  average=`+(@gauged_times)/(float)number_of_times;
  gauged_times=sort(gauged_times);

  write("average: %0, min: %0, max: %0                              \n",
```

```
        average, gauged_times[0], gauged_times[-1]);
}
```

Short Sleeps

```
int abort_on_signal=1;          // if true, aport on signal

sleep(0.25, abort_on_signal);

delay(0.25); // uses busy-wait for accuracy,

             // may be interrupted by signal handlers
```

Program: hopdelta

```
Calendar.dwim_time("Tue, 26 May 1998 23:57:38 -0400")->distance(
    Calendar.dwim_time("Wed, 27 May 1998 05:04:03 +0100"))->format_elapsed
// Result: "0:06:25"


// download the following standalone program
#!/usr/bin/pike
// chapter 3.11
// hopdelta - feed mail header, produce lines
//            showing delay at each hop.
int main()
{
  MIME.Message        mail = MIME.Message(Stdio.stdin.read());

  array           received = reverse(mail->headers->received/"\0");
  Calendar.Second lasttime = Calendar.dwim_time(mail->headers->date);

  array delays=({ ({ "Sender", "Recipient", "Time", "Delta" }) });
  delays+=({ ({ mail->headers->from,
                array_sscanf(received[0], "from %[^ ]")[0],
                mail->headers->date,
                ""
          }) });


  foreach(received;; string hop)
  {
    string fromby, date;
    [fromby, date] = hop/";";
    Calendar.Second thistime = Calendar.dwim_time(date);
```

```
    delays+= ({ array_sscanf(fromby, "from %[^ ]%*sby %[^ ]%*s") +
                ({ date, lasttime->distance(thistime)->format_elapsed() })
             });

    lasttime=thistime;
  }

  write("%{%-=22s %-=22s %-=20s %=10s\n%}\n", delays);
  return 0;
}
```

---

arrays.html

PLEAC-Pike

---

4. Arrays

Introduction

```
// nested arrays are supported

array flat = ({ "this", "that", "the", "other" });
array nested = ({ "this", "that", ({ "the", "other" }) });


array tune = ({ "The", "Star-Spangled", "Banner" });
tune[0];
// Result: "The"

tune[1];
// Result: "Star-Spangled"



// the typing may be more specific

// only strings allowed in the array (thus no nesting!)
```

```
array(string) flat = ({ "this", "that", "the", "other" });

// allow one level of nesting

array(string|array(string)) admit1 = ({ "this", "that", ({ "the", "other"

// the first level may only contain arrays, other levels may contain anyth

array(array) require1 ({ ({ "this", "that" }), ({ "the", "other" }) });
```

Specifying a List In Your Program

```
// list

array(string) a = ({ "quick", "brown", "fox" });

// words

array(string) a = "Why are you teasing me?"/" ";

// lines

array(string) lines = #"The boy stood on the burning deck,
It was as hot as glass."/"\n";

// file

array(string) bigarray = Stdio.read_file("mydatafile")/"\n";

// the quoting issues do not apply.


array(string) ships = "Niña Pinta Santa María"/" ";         // wrong

array(string) ships = ({ "Niña", "Pinta", "Santa María" }); // right
```

Printing a List with Commas

```
// download the following standalone program
#!/usr/bin/pike
// chapter 4.2
// commify_series - show proper comma insertion in list output
```

```
array(array(string)) lists =
  ({
    ({ "just one thing" }),
    ({ "Mutt", "Jeff" }),
    ({ "Peter", "Paul", "Mary" }),
    ({ "To our parents", "Mother Theresa", "God" }),
    ({ "pastrami", "ham and cheese", "peanut butter and jelly", "tuna" }),
    ({ "recycle tired, old phrases", "ponder big, happy thoughts" }),
    ({ "recycle tired, old phrases",
       "ponder big, happy thoughts",
       "sleep and dream peacefully" }),
  });

void main()
{
  write("The list is: %s.\n", commify_list(lists[*])[*]);
}

string commify_list(array(string) list)
{
  switch(sizeof(list))
  {
    case 1: return list[0];
    case 2: return sprintf("%s and %s", @list);
    default:
      string seperator=",";
      int count;
      while(count<sizeof(list) && search(list[count], seperator)==-1)
        count++;
      if(count<sizeof(list))
        seperator=";";
      return sprintf("%{%s"+seperator+" %}and %s",
                     list[..sizeof(list)-2], list[-1]);
  }
}
```

Changing Array Size

```
void what_about_that_array(array list)
{
    write("The array now has %d elements.\n", sizeof(list));
    write("The index of the last element is %d.\n", sizeof(list)-1);
    write("Element #3 is %O.\n", list[3]);
}

array people = ({ "Crosby", "Stills", "Nash", "Young" });
what_about_that_array(people);
```

```
// The array now has 4 elements.

// The index of the last element is 3.

// Element #3 is "Young".


people=people[..sizeof(people)-2];
what_about_that_array(people);
// The array now has 3 elements.

// The index of the last element is 2.

// Index 3 is out of array range -3..2.


people+=allocate(10001-sizeof(people));
what_about_that_array(people);
// The array now has 10001 elements.

// The index of the last element is 10000.

// Element #3 is 0.


array people = ({ "Crosby", "Stills", "Nash", "Young" }); // resetting the

people[10000]=0;
// Index 10000 is out of array range -4..3.

// accessing a nonexisting index is always an error.

// arrays can not be enlarged this way.
```

Doing Something with Every Element in a List

```
foreach(list; int index; mixed item)
{
  // do something with item (and possibly index)

}

foreach(bad_users;; object user)
{
  complain(user);
}
```

```
// for such simple cases pike provides a convenient automap feature:

complain(bad_users[*]);
// will do the same as the foreach above.


foreach(sort(indices(getenv()));; string var)
{
  write("%s=%s\n", var, getenv(var));
}

// if you don't need an assurance that the indices are sorted (they most l

// are sorted anyways) you may use:

foreach(getenv(); string var; string value)
{
  write("%s=%s\n", var, value);
}

foreach(all_users;; string user)
{
  int disk_space = get_usage(user);
  if(disk_space > MAX_QUOTA)
    complain(user);
}

// continue; to jump to the next

// break; to stop the loop

// redo can be done by doing a loop with the proper checks in the block


object pipe=Stdio.File();
Process.create_process(({ "who" }), ([ "stdout":pipe->pipe() ]));
foreach(pipe->line_iterator();; string line)
{
  if(search(line, "tchrist")>-1)
    write(line+"\n");
}

object fh=Stdio.File("somefile");
foreach(fh->line_iterator(); int linenr; string line)
{
  foreach(Process.split_quoted_string(line);; string word)//split on white

  {
    write(reverse(word));
  }
}
```

```
array(int) list = ({ 1,2,3 });
foreach(list;; int item)
{
  item--;
}
write("%{%d %}\n", list);
// Result: 1 2 3


// we can still use foreach instead of for,

// because foreach gives us the index as well:

foreach(list; int index;)
{
  list[index]--;
}
write("%{%d %}\n", list);
// Result: 0 1 2


array a = ({ 0.5, 3 });
array b = ({ 0, 1 });
// foreach handles only one array so there is nothing to gain here.

// better use automap:

array a_ = a[*]*7;
array b_ = b[*]*7;
write("%{%O %}\n", a_+b_);
// 3.500000 21 0 7


string scalar = " abc ";
array(string) list = ({ " a ", " b " });
mapping(mixed:string) hash = ([ "a":" a ", "b":" b " ]);

scalar = String.trim_whites(scalar);
list = String.trim_whites(list[*]);
foreach(hash; int key;)
{
  hash[key]=String.trim_whites(hash[key]);
}
```

Iterating Over an Array by Reference

```
// pike does not distinguish between arrays and array references
```

```
// (they are all references anyways) so this section does not apply
```

Extracting Unique Elements from a List

```
mapping seen = ([]);
array   uniq = ({});
foreach(list;; mixed item)
{
  if(!seen[item])
    seen[item] = 1;
  else
    uniq += ({ item });
}
```

```
mapping seen = ([]);
array   uniq = ({});
foreach(list;; mixed item)
{
  if(!seen[item]++)
    uniq += ({ item });
}
```

```
mapping seen = ([]);
array   uniq = ({});
foreach(list;; mixed item)
{
  if(!seen[item]++)
    some_func(item);
}
```

```
// the following is probably the most natural for pike
```

```
mapping seen = ([]);
array   uniq = ({});
foreach(list;; mixed item)
{
  seen[item]++;
}
uniq = indices(seen);
```

```
// not necessarily faster but shorter:
```

```
array uniq = indices(({ list[*],1 }));
```

```
// also short, and preserving the originaal order:
```

```
array uniq = list&indices(({ list[*],1 }));


object pipe = Stdio.File();
Process.create_process(({ "who" }), ([ "stdout":pipe->pipe() ]));
mapping ucnt = ([]);
foreach(pipe->line_iterator();; string line)
{
  ucnt[(line/" ")[0]]++;
}

array users = sort(indices(ucnt));
write("users logged in: %s\n", users*" ");
```

Finding Elements in One Array but Not Another

```
// one of pikes strenghts are operators.

// the following are the only idiomatic solutions to the problem


array A = ({ 1, 2, 3 });
array B = ({ 2, 3, 4 });
array aonly = A-B;
// Result: ({ 1 });
```

Computing Union, Intersection, or Difference of Unique Lists

```
array a = ({ 1, 3, 5, 6, 7, 8 });
array b = ({ 2, 3, 5, 7, 9 });

// union:

array union = a|b;
// ({ 1, 3, 5, 6, 7, 8, 2, 9 })


// intersection

array intersection = a&b;
// ({ 3, 5, 7 })
```

```
// difference

array difference = a-b;
// ({ 1, 6, 8 })


// symetric difference

array symdiff= a^b;
// ({ 1, 6, 8, 2, 9 })
```

Appending One Array to Another

```
// join arrays


// appending to an array will always create a new array and pike is desig

// handle this efficiently.


array members = ({ "Time", "Flies" });
array initiates = ({ "An", "Arrow" });
members += initiates;
// members is now ({ "Time", "Flies", "An", "Arrow" })


members = members[..1]+({ "Like" })+members[2..];
write("%s\n", members*" ");

members[0] = "Fruit";
members = members[..sizeof(members)-3]+({ "A", "Banana" });
write("%s\n", members*" ");

// Time Flies Like An Arrow

// Fruit Flies Like A Banana
```

Reversing an Array

```
// almost any operation you do on the elements will add more overhead thar
```

```
// reversing the array, if there is any possible optimization, pike will c

// for you.


array reversed = reverse(arr);

// unless you were going to use for anyways then foreach(reverse( ...)) is

// preferable.

foreach(reverse(arr);; mixed item)
{
  // do something with item

}

for(int i=sizeof(arr)-1; i<=0; i--)
{
  // so something with arr[i]

}

array ascending = sort(users);
array descending = reverse(sort(users));

// reverse(sort()) is faster by a magnitude

array descending = Array.sort_array(users, lambda(mixed a, mixed b)
                                           {
                                             return a<b;
                                           }
                                    );
```

Processing Multiple Elements of an Array

```
array arr = ({ 0,1,2,3,4,5,6,7,8,9 });
int n=3;
array front = arr[..n-1];
arr = arr[n..];

array back = arr[sizeof(arr)-n..];
arr = arr[..sizeof(arr)-(n+1)];

// since new arrays are created if elements are added or removed

// shift and pop are not usefull here.
```

```
// if you need shift and pop capabilities use the ADT classes:


array shift2(ADT.Queue queue)
{
  return ({ queue->read(), queue->read() });
}

ADT.Queue friends = ADT.Queue("Peter", "Paul", "Mary", "Jim", "Tim");
string this, that;
[this, that] = shift2(friends);
// this contains Peter, that has Paul, and

// friends has Mary, Jim, and Tim


ADT.Stack beverages = ADT.Stack();
beverages->set_stack(({ "Dew", "Jolt", "Cola", "Sprite", "Fresca" }));
array pair = beverages->pop(2); // implementing pop2 would gain nothing he

// pair[0] contains Sprite, pair[1] has Fresca,

// and beverages has (Dew, Jolt, Cola)



// to be able to shift and pop on the same list use the following:


array shift2(ADT.CircularList list)
{
  return ({ list->pop_front(), list->pop_front() });
}

array pop2(ADT.CircularList list)
{
  return reverse( ({ list->pop_back(), list->pop_back() }) );
}

ADT.CircularList friends = ADT.CircularList( ({"Peter", "Paul", "Mary", "J
string this, that;
[this, that] = shift2(friends);
// this contains Peter, that has Paul, and

// friends has Mary, Jim, and Tim


ADT.CircularList beverages = ADT.CircularList( ({ "Dew", "Jolt", "Cola", "
array pair = pop2(beverates);
// pair[0] contains Sprite, pair[1] has Fresca,
```

```
// and beverages has (Dew, Jolt, Cola)
```

Finding the First List Element That Passes a Test

```
mixed match = search(arr, element);

int test(mixed element)
{
  if(sizeof(element)==5)
    return 1;
  else
    return 0;
}

mixed match = Array.search_array(arr, test);

if(match != -1)
{
  // do something with arr[match]

}
else
{
  // do something else

}

// another convenient way if you do many tests on the same list,

// and you do not care for the position is:


if( (multiset)arr[element] )
{
  // found

}
else
{
  // not found

}
```

Finding All Elements in an Array Matching Certain Criteria

```
array matching=({});

foreach(list;; mixed element)
{
  if(test(element))
    matching+=({ element });
}

array matching = map(list, test)-({ 0 });
array matching = test(list[*])-({ 0 });
// apply test() on each element in list, collect the results, and remove

// results that are 0.
```

Sorting an Array Numerically

```
// since pike has different types for strings and numbers, ints and floats

// of course sorted numerically

// (sort() is destructive, the original array is changed)


array(int) unsorted = ...;
array(int) sorted = sort(unsorted);


// but suppose you want to sort an array of strings by their numeric value

// things get a bit more interresting:


array(string) unsorted = ({ "123asdf", "3poiu", "23qwert", "3ayxcv" });

sort((array(int))unsorted, unsorted);
// unsorted is now sorted.
```

Sorting a List by Computable Field

```
array unordered;
int compare(mixed a, mixed b)
{
  // return comparison of a and b

}
array ordered = Array.sort_array(unordered, compare);

//---------------------------------------------------------------

int compute(mixed element)
{
  // return computation from element

}
array precomputed = map(unordered, compute);
sort(precomputed, unordered); // will destructively sort unordered in the

array ordered = unordered;    // manner as precomputed.


//---------------------------------------------------------------

sort(map(unordered, compute), unordered); // without a temp variable

sort(compute(unordered[*]), unordered);   // using the automap operator

                                          // both get compiled to the same



//---------------------------------------------------------------

array ordered = sort(employees, lambda(mixed a, mixed b)
                                {
                                  return a->name > b->name;
                                }
                    );

//---------------------------------------------------------------

foreach(Array.sort_array(employees,
                         lambda(mixed a, mixed b){ return a->name > b->nam
        ;; mixed employee)
{
  write("%s earns $%d\n", employee->name, employee->salary);
}
```

```
//-------------------------------------------------------------

array ordered_employees =
        Array.sort_array(employees,
                         lambda(mixed a, mixed b){ return a->name > b->nam
foreach(ordered_employees;; mixed employee)
{
  write("%s earns $%d\n", employee->name, employee->salary);
}

mapping bonus;
foreach(ordered_employees;; mixed employee)
{
  // you are not supposed to use the social security number as an id

  if(bonus[employee->id])
    write("%s got a bonus!\n", employee->name);
}

//-------------------------------------------------------------

array sorted = Array.sort_array(employees,
                                lambda(mixed a, mixed b)
                                {
                                  if(a->name!=b->name)
                                    return (a->name < b->name)
                                  return (b->age < a->age);
                                }
                               );


//-------------------------------------------------------------

array(array) users = System.get_all_users();

sort(users);
// System.get_all_users() returns an array of arrays, with the name as the

// first element in each inner array, sort handles multidimensional arrays

// we can skip creating our own sort function.


// if we wanted to sort on something else one could rearrange the array:

array user;
while(user=System.getpwent())
{
  users += ({ user[2], user });
}
System.endpwent();
sort(users);  // now we are sorting by uid.
```

```
// alternative:

array(array) users = System.get_all_users();
sort(users[*][2], users);

write(users[*][0]*"\n");
write("\n");

//------------------------------------------------------------------

array names;
array sorted = Array.sort_array(names, lambda(mixed a, mixed b)
                                       {
                                         return a[1] < b[1];
                                       }
                                );
// faster:

sort(names[*][1], names);
sorted=names;
//------------------------------------------------------------------

array strings;
array sorted = Array.sort_array(strings, lambda(mixed a, mixed b)
                                       {
                                         return sizeof(a) < sizeof(b);
                                       }
                                );
// faster:

sort(sizeof(strings[*]), strings);
sorted=strings;
//------------------------------------------------------------------

array strings;
array temp = map(strings, sizeof);
sort(temp, strings);
array sorted = strings;
//------------------------------------------------------------------

array strings;
sort(map(strings, sizeof), strings);   // pick one

sort(sizeof(strings[*]), strings);
sorted=strings;
//------------------------------------------------------------------

array fields;
array temp = map(fields, array_sscanf, "%*s%d%*s");
sort(temp, fields);
array sorted_fields=fields;
```

```
//-----------------------------------------------------------------

sort(array_sscanf(fields[*], "%*s%d%*s"), fields);
array sorted_fields=fields;
//-----------------------------------------------------------------

array passwd_lines = (Stdio.read_file("/etc/passwd")/"\n")-({""});
array(array) passwd = passwd_lines[*]/":";

int compare(mixed a, mixed b)
{
  if(a[3]!=b[3])
    return (int)a[3]<(int)b[3];
  if(a[2]!=b[2])
    return (int)a[2]<(int)b[2];
  return a[0]<b[0];
}

array sorted_passwd = Array.sort_array(passwd, compare);

// alternatively the following uses the builtin sort

sort( passwd[*][0], passwd);
sort( ((array(int))passwd[*][2]), passwd);
sort( ((array(int))passwd[*][3]), passwd);
```

Implementing a Circular List

```
ADT.CircularList circular;
circular->push_front(circular->pop_back());
circular->push_back(circular->pop_front());
//-----------------------------------------------------------------

mixed grab_and_rotate(ADT.CircularList list)
{
  mixed element = list->pop_front();
  list->push_back(element);
  return element;
}

ADT.CircularList processes = ADT.CircularList( ({ 1, 2, 3, 4, 5 }) );
while(1)
{
  int process = grab_and_rotate(processes);
  write("Handling process %d\n", process);
  sleep(1);
}
```

Randomizing an Array

```
array arr;
Array.shuffle(arr);  // this uses the fisher-yates shuffle



//-----------------------------------------------------------------


// being creative with the algorithm, this is not as memory efficient,

// but it shows the utility of multisets.

array set_shuffle(array list)
{
  multiset elements=(multiset)list;
  list=({});                      // reset the list

  while(sizeof(elements))         // while we still have elements left

  {
    mixed pick=random(elements); // pick a random element

    list+=({ pick });             // add it to the new list

    elements[pick]--;             // remove the element we picked

  }
  return list;
}

array list;
list=set_shuffle(list);

//-----------------------------------------------------------------


inherit "mjd_permute";
int permutations = factorial(sizeof(list));
array shuffle = list[n2perm(random(permutations)+1, sizeof(list))[*]];

//-----------------------------------------------------------------


void naive_shuffle(array list)
{
  for(int i=0; i<sizeof(list); i++)
  {
    int j=random(sizeof(list)-1);
```

```
    [ list[i], list[j] ] = ({ list[j], list[i] });
  }
}
```

Program: words

```
// download the following standalone program
#!/usr/bin/pike
// section 4.18 example 4.2
// words - gather lines, present in columns

void main()
{
  array words=Stdio.stdin.read()/"\n";   // get all input
  int maxlen=sort(sizeof(words[*]))[-1]; // sort by size and pick the larg
  maxlen++;                              // add space

  // get boundaries, this should be portable
  int cols = Stdio.stdout->tcgetattr()->columns/maxlen;
  int rows = (sizeof(words)/cols) + 1;

  string mask="%{%-"+maxlen+"s%}\n";     // compute format

  words=Array.transpose(words/rows);     // split into groups as large as
                                         // number of rows and then transp
  write(mask, words[*]);                 // apply mask to each group
}
```

Program: permute

```
int factorial(int n)
{
  int s=1;
  while(n)
    s*=n--;
  return s;
}
write("%d\n", factorial(500));
// TODO: provide a short example using Array.permute()

//-----------------------------------------------------------------

// download the following standalone program
#!/usr/bin/pike
```

```
void main()
{
  string line;
  while(line=Stdio.stdin->gets())
  {
    permute(line/" ");
  }
}

void permute(array items, array|void perms)
{
  if(!perms)
    perms=({});
  if(!sizeof(items))
    write((perms*" ")+"\n");
  else
  {
    foreach(items; int i;)
    {
      array newitems=items[..i-1]+items[i+1..];
      array newperms=items[i..i]+perms;
      permute(newitems, newperms);
    }
  }
}

//----------------------------------------------------------------

// download the following standalone program
#!/usr/bin/pike

mapping fact=([ 1:1 ]);

int factorial(int n)
{
  if(!fact[n])
    fact[n]=n*factorial(n-1);
  return fact[n];
}

array n2pat(int N, int len)
{
  int i=1;
  array pat=({});

  while(i <= len)
  {
    pat += ({ N%i });
    N/=i;
    i++;
  }
```

```
  return pat;
}

array pat2perm(array pat)
{
  array source=indices(pat);
  array perm=({});
  while(sizeof(pat))
  {
    perm += ({ source[pat[-1]] });
    source = source[..pat[-1]-1]+source[pat[-1]+1..];
    pat=pat[..sizeof(pat)-2];
  }
  return perm;
}

array n2perm(int N, int len)
{
  return pat2perm(n2pat(N, len));
}

void main()
{
  array data;
  while(data=Stdio.stdin->gets()/" ")
  {
    int num_permutations = factorial(sizeof(data));
    for(int i; i<num_permutations; i++)
    {
      array permutation = data[n2perm(i, sizeof(data))[*]];
      write(permutation*" "+"\n");
    }
  }
}
```


Prev Home Next

Dates and Times Hashes

hashes.html

PLEAC-Pike

Prev Next

5. Hashes

Introduction

```
// creating a mapping from arrays

mapping age = mkmapping( ({ "Nat", "Jules", "Josh", }), ({ 24, 25, 17 }) )

// initialize one index at a time

mapping age = ([]);
age["Nat"] = 24;
age["Jules"] = 25;
age["Josh"] = 17;

// if your index names are valid identifiers:

age->Nat = 24;
age->Jules = 25;
age->Josh = 17;

// the traditional way to initialize mappings

mapping age = ([ "Nat":24, "Jules":25, "Josh":17 ]);

mapping(string:string) food_color = ([
                                      "Apple":"red",
                                      "Banana":"yellow",
                                      "Lemon":"yellow",
                                      "Carrot":"orange"
                                     ]);

// a index may be of any type

mapping any = ([ "1":"a string", 1:"an int", 1.0:"a float" ]);

// you may use other types too, but be aware that they are matched by

// reference, and not by value.
```

Adding an Element to a Hash

```
mapping[mixed] = mixed;
mapping->string = mixed;  //any string that is a valid identifier
```

```
// food_color as per section 5.0

food_color->Raspberry = "pink";
write("Known foods:\n");
foreach(food_color; string food; )
{
  write(food+"\n");
}
// Lemon

// Banana

// Apple

// Carrot

// Raspberry
```

Testing for the Presence of a Key in a Hash

```
// an undefined value in a mapping gets turned to 0.

// assigning 0 as a value is allowed and will not remove the index.

// checking for the index will of course return 0 and be interpreted as fa

// to check if the index is really there, use zero_type()


if(!zero_type(mapping->index))
{
  // it exists

}
else
{
  // it doesn't

}

// food_color as per section 5.0

foreach( ({ "Banana", "Milk" }) ;; string name)
{
  if(!zero_type(food_color[name]))
    write("%s is a food.\n", name);
```

```
  else
    write("%s is a drink.\n", name);
}
// Banana is a food.

// Milk is a drink.


// ------------------------------------------------------------

mapping age = ([ "Toddler":3,
                 "Unborn":0,
                 "Newborn":0.0,
                 "Phantasm":UNDEFINED ]);

foreach( ({ "Toddler", "Unborn", "Newborn", "Phantasm", "Relic" });; strin
{
    write(thing+":");
    if(!zero_type(age[thing]))
      write(" Exists");
    if(age[thing])
      write(" True");
    write("\n");
}
// Toddler: Exists True

// Unborn: Exists

// Newborn: Exists True

// Phantasm: Exists

// Relic:


// age->Toddler exists, because zero_type() is only true if the index is n

// the mapping. it is true because the value is not 0.

// age->Unborn exists, but is false because 0 is false

// age->Newborn exists and is true, because 0.0 is not false

// age->Phantasm exists and is false, like Unborn

// age->Relic does not exist


// we can not test for defined. UNDEFINED is a special value used internal

// the compiler. it gets converted to 0 as soon as it is assigned in a map
```

```
// however we can create something equivalent that can be treated like any

// other value, except that it is false:


class Nil
{
  // this is a minimal example.

  // a more complete one would also handle casting


  int `!() {return 1;}
  string _sprintf() {return "Nil";}

  // we could have this function externally, but this is more convenient

  int defined(mixed var)
  {
    return !zero_type(var) && var!=this;
  }
}

Nil NIL = Nil();                    // create an instance so we can use it

function defined = NIL->defined;  // just for symetry


mapping age = ([ "Toddler":3,
                 "Unborn":0,
                 "Phantasm":NIL ]);

foreach( ({ "Toddler", "Unborn", "Phantasm", "Relic" });; string thing)
{
    write(thing+":");
    if(!zero_type(age[thing]))
      write(" Exists");
    if(defined(age[thing]))
      write(" Defined");
    if(age[thing])
      write(" True");
    write("\n");
}

// Toddler: Exists Defined True

// Unborn: Exists Defined

// Phantasm: Exists

// Relic:
```

```
// age->Toddler exists, because zero_type() is only true if the index is n
// the mapping. it is defined because it exists an is not NIL.
// it is true because the value is not 0.
// age->Unborn exists, and is defined, but is false because 0 is false
// age->Phantasm exists, is not defined because it is NIL,
// it is not true because NIL is false.
// age->Relic does not exist, it is not defined even though it is not NIL
// because it doesn't exist. it is also not true, because it does not exis

// ------------------------------------------------------

mapping size = ([]);
string filename;
while(filename = Stdio.stdin->gets())
{
  filename -= "\n";
  if(size[filename])                // wrong

    continue;
  if(!zero_type(size[filename]))    // right

    continue
  object stat = file_stat(filename)
  if(stat)
    size[filename] = stat->size;
  else
    size[filename] = -1; // since sizes can't be negative, this will do.
                         // if -1 is a valid value, use NIL
}
```

Deleting from a Hash

```
// users occasionally may get the idea that mapping[index]=0; may remove i
// from mapping. the normal way to remove a index from a mapping is to
// subtract: mapping -= ([ index:0 ]); the following shall demonstrate the
```

```
// difference between subtracting a index and assigning 0 to it.


// food_color as per section 5.0

void print_foods()
{
  write("Foods:%{ %s%}\n", indices(food_color));
  write("Values: ");

  foreach(food_color; string food; string color)
  {
    if(color)
      write(color+" ");
    else
      write("(no value) ");
  }
  write("\n");
}

write("Initially:\n");
print_foods();

write("\nWith Banana set to 0\n");
food_color->Banana = 0;
print_foods();

write("\nWith Banana deleted\n");
food_color -= ([ "Banana":"the value is irrelevant" ]);
print_foods();

// Initially:

// Foods: Lemon Banana Apple Carrot

// Values: yellow yellow red orange

//

// With Banana set to 0

// Foods: Banana Lemon Apple Carrot

// Values: (no value) yellow red orange

//

// With Banana deleted

// Foods: Lemon Carrot Apple
```

```
// Values: yellow orange red

// you can also subtract multiple indices:
food_color -= ([ "Banana":0, "Apple":0, "Cabbage":0 ]);
// note that subtracting a mapping from another creates a new mapping.
// thus any references you have to a mapping will be broken.
// in most cases this is what you want anyways. if it is not, you can also
// remove indices using m_delete();

m_delete(food_color, "Banana");
```

Traversing a Hash

```
foreach( mapping; type index; type value)
{
  //do something with index and value

}

// food_color as per 5.0
foreach(food_color; string food; string color)
{
  write("%s is %s.\n", food, color);
}

// Banana is yellow.
// Lemon is yellow.
// Carrot is orange.
// Apple is red.


foreach(sort(indices(food_color));; string food)
{
  write("%s is %s.\n", food, food_color[food]);
}
```

```
// Apple is red.

// Banana is yellow.

// Carrot is orange.

// Lemon is yellow.


// since pike does not have any equivalent to each() its problems do not a


// download the following standalone program
#!/usr/bin/pike
// countfrom - count number of messages from each sender

void main(int argc, array argv)
{
  object file;
  mapping from = ([]);

  if(sizeof(argv)>1)
    file = Stdio.File(argv[1], "r");
  else
    file = Stdio.stdin;


  foreach(file; int count; string line)
  {
    array email = array_sscanf(line, "From: %s");
    if(sizeof(email))
      from[email[0]]++;
  }
  write("end\n");

  foreach(sort(indices(from));; string person)
  {
    write("%s: %d\n", person, from[person]);
  }
}
```

Printing a Hash

```
// perls problems and solutions do not apply to pike

// here are a few ways to print a mapping:

// food_color as per 5.0
```

```
// debugging style

write("%O\n", food_color);
// ([ /* 4 elements */

//   "Apple": "red",

//   "Banana": "yellow",

//   "Carrot": "orange",

//   "Lemon": "yellow"

// ])


// one element at a time:

foreach(food_color; string food; string color)
{
  write("%s is %s\n", food, color);
}
// Lemon is yellow

// Carrot is orange

// Banana is yellow

// Apple is red


// with the help of an array

write("%{%s is %s\n%}", sort((array)food_color));
// Apple is red

// Banana is yellow

// Carrot is orange

// Lemon is yellow
```

Retrieving from a Hash in Insertion Order

```
// for this we need to first create an OrderedMapping class.
```

```
// work for this is in progress

// @@INCOMPLETE@@

// @@INCOMPLETE@@
```

Hashes with Multiple Values Per Key

```
mapping(string:array(string)) ttys = ([]);

object pipe = Stdio.File();
Process.create_process(({ "who" }), ([ "stdout":pipe->pipe() ]));

foreach(pipe->line_iterator();; string line)
{
  array tty=(line/" ")-({ "" });
  if(!ttys[tty[0]])
    ttys[tty[0]] = ({ tty[1] });
  else
    ttys[tty[0]] += ({ tty[1] });
}

foreach(sort(indices(ttys));; string user)
{
  write("%s: %{%s %}\n", user, ttys[user]);
}

foreach(sort(indices(ttys));; string user)
{
  write("%s: %d ttys.\n", user, sizeof(ttys[user]));
  foreach(ttys[user];; string tty)
  {
    object stat = file_stat("/dev/"+tty);
    string user;
    if(stat)
      user = getpwuid(stat->uid)[0];
    else
      user = "(not available)";
    write("\t%s (owned by %s)\n", tty, user);
  }
}

mapping multihash_delete(mapping hash, mixed key, mixed value)
{
  if(arrayp(hash[key]))
    hash[key]-=({ value });
  if(!sizeof(hash[key]))
    m_delete(hash, key);
```

```
  return hash;
}
```

Inverting a Hash

```
// search for a value in a mapping

mapping lookup;
mixed value;
mixed key = search(lookup, value);

// transposing: (this will break if there are multiple occurances of a val

mapping lookup;
mapping reverse = mkmapping(values(lookup), indices(lookup));
//----------------------------------------------------------

mapping surname = ([ "Mickey":"Mantle", "Babe":"Ruth" ]);
write("%s\n", search(surname, "Mantle"));
// Mikey


// with a transposed mapping (only worth doing if you'd have to search a l

mapping first_name = mkmapping(values(surname), indices(surname));
write("%s\n", first_name->Mantle);
// Mikey

//----------------------------------------------------------

// download the following standalone program
#!/usr/bin/pike

void main(int argc, array(string) argv)
{
  if(argc < 2)
  {
    write("usage: foodfind food_or_color\n");
  }

  string given = argv[1];
  mapping color = ([ "Apple":"red",
                     "Banana":"yellow",
                     "Lemon":"yellow",
                     "Carrot":"orange"
                  ]);
  if(color[given])
    write("%s is a food with color %s\n", given, color[given]);
```

```
  string food = search(color, given);
  if(food)
    write("%s is a food with color %s\n", food, given);

// search will only find one value,
// but it can be given a place where it should start searching

  array foods = ({ search(color, given) });
  food = 0;
  while(food = search(color, given, foods[-1]))
  {
    foods += ({ food });
  }
  write("%{%s %}were %s foods.\n", foods, given);

}

//-------------------------------------------------------------


// food_color as per 5.0

mapping foods_with_color = ([]);
foreach(food_color; string food; string color)
{
  if(!foods_with_color[color])
    foods_with_color[color] = ({ food });
  else
    foods_with_color[color] += ({ food });
}

write("%{%s %}were yellow foods.\n", foods_with_color->yellow);
```

Sorting a Hash

```
mapping hash;
foreach(sort(indices(hash));; mixed key)
{
  mixed value = hash[key];
  // do something with key, value

}

foreach(sort(indices(food_color));; string food)
{
  write("%s is %s.\n", food, food_color[food]);
}

array foods = indices(food_color);
```

```
sort(sizeof(values(food_color)[*]), foods);

foreach(foods;; string food)
{
  write("%s is %s.\n", food, food_color[food]);
}
```

Merging Hashes

```
// the natural way to merge two mappings is to use + or |

// for mappings both operations are equivalent.

mapping A, B;
mapping merged = A + B;
mapping merged = A | B;

// if an index is in both mappings, the value will be taken from the secon

// one.


mapping drink_color = ([ "Milk":"white",
                         "Tomato juice":"red" ]);

mapping ingested_color = drink_color + food_color;

//  Result: ([ /* 6 elements */

//             "Apple": "red",

//             "Banana": "yellow",

//             "Carrot": "orange",

//             "Lemon": "yellow",

//             "Milk": "white",

//             "Tomato juice": "red"

//           ])
```

Finding Common or Different Keys in Two Hashes

```
// create a mapping where indices are in both A and B

mapping both = A & B;

// in A or B, but not in both

mapping one = A ^ B;

// in A, but not in B

mapping exA = A - B;

mapping citrus_color = ([ "Lemon":"yellow",
                          "Orange":"orange",
                          "Lime":"green"
                        ]);

array non_citrus = indices(food_color - citrus_color);
```

Hashing References

```
// this problem does not apply to pike

// any value may be used as an index, if you get an object reference from

// anywhere, if will work, if the index in the mapping is actually the sam

// object.

// however note that the same value is not the same reference:

array a = ({ 1,2 });
array b = ({ 1,2 });
mapping m = ([ a:"a" ]);
m[b];
// Result: 0 (b will not be found.)

m[b]="b";
m;
// Result: ([ ({ 1, 2 }): "a",

//           ({ 1, 2 }): "b"
```

```
//           ])
// this looks as if the mapping has the same index twice
// but since they are references this is not the case.
```

Presizing a Hash

```
// this problem does not apply to pike
// pike uses a smart preallocation algorythm that will avoid the need to
// allocate memory everytime an element is added
```

Finding the Most Common Anything

```
mapping count = ([]);
foreach(ARRAY;; mixed element)
{
    count[element]++;
}
```

Representing Relationships Between Data

```
mapping father = ([ "Cain":"Adam",
                    "Abel":"Adam",
                    "Seth":"Adam",
                    "Enoch":"Cain",
                    "Irad":"Enoch",
                    "Mehujael":"Irad",
                    "Methusael":"Mehujael",
                    "Lamech":"Methusael",
                    "Jabal":"Lamech",
                    "Jubal":"Lamech",
                    "Tubalcain":"Lamech",
                    "Enos":"Seth"
                 ]);

foreach(Stdio.stdin;; string name)
{
  do
```

```
  {
    write("%s ", name);
  } while(name = father[name]);
  write("\n");
}

mapping children = ([]);
foreach(father; string k; string v)
{
  if(!children[v])
    children[v] = ({ k });
  else
    children[v] += ({ k });
}

foreach(Stdio.stdin;; string name)
{
  write("%s begat %s.\n", name, (children[name]||({ "nobody" }))*", ");
}

//-----------------------------------------------------


mapping includes = ([]);
foreach(files, string file)
{
  string F = Stdio.read_file(file);
  if(!F)
    werror("Couldn't read %s; skipping.\n", file);

  foreach(F/"\n";; string line)
  {
    array included = array_sscanf(line, "%*[ \t]#include%*[ \t]%*[<\"]%s%*
    if(sizeof(included))
    {
      if(!includes[included[0]])
        includes[included[0]] = ({ file });
      else
        includes[included[0]] += ({ file });
    }
  }
}
//-----------------------------------------------------


array uniq = `|( @values(includes) );
array include_free = sort( uniq - indices(includes) );

// values(includes) is an array of arrays.

// @ splices values(includes) as arguments into `|()
```

```
// `|() is a function that represents the | operator: a|b|c is `|(a,b,c)

// | on arrays creates a new array with elements in a or b

// the resulting array is unique as long as each array only has unique ele

// from the uniq array we remove all those that are used as indices in the

// includes mapping.

// at last we sort the remaining list.
```

Program: dutree

```
// download the following standalone program
#!/usr/bin/pike
// dutree - print sorted indented rendition of du output
// as a slight deviation from the perl version, the sizes are still in one
// column, to make it more readable.

array input(array(string) args)
{
  mapping Dirsize=([]);
  mapping Kids=([]);

  object pipe = Stdio.File();
  Process.create_process(({ "du" })+args, ([ "stdout":pipe->pipe() ]));

  string name;
  foreach(pipe->line_iterator();; string line)
  {
    int size;
    [size, name] = array_sscanf(line, "%d%*[ \t]%s");
    Dirsize[name] = size;
    array path = name/"/";
    string parent = path[..sizeof(path)-2]*"/";
    if(!Kids[parent])
      Kids[parent] = ({ name });
    else
      Kids[parent] += ({ name });
  }
  return ({ name, Dirsize, Kids });
}

void getdots(string root, mapping Dirsize, mapping Kids)
{
  int size, cursize;
  size = cursize = Dirsize[root];
```

```
  if(Kids[root])
  {
    foreach(Kids[root];; string kid)
    {
      cursize -= Dirsize[kid];
      getdots(kid, Dirsize, Kids);
    }
  }
  else
    Kids[root] = ({});

  if(size != cursize)
  {
    string dot = root+"/.";
    Dirsize[dot] = cursize;
    Kids[root] += ({ dot });
  }
}

void output(string root, mapping Dirsize, mapping Kids,
            int width, void|string prefix)
{
  if(!prefix)
    prefix="";
  string path = (root/"/")[-1];
  int size = Dirsize[root];
  write("%*d %s%s\n", width, size, prefix, path);
  prefix += "|" + " "*(sizeof(path)-1);
  if(Kids[root])
  {
    array kids = Kids[root];
    // get the dirsize for each kid and sort by that
    sort(Dirsize[kids[*]], kids);

    // make the output for each kid
    output(kids[*], Dirsize, Kids, width, prefix);
  }
}

void main(int argc, array(string) argv)
{
  mapping Dirsize;
  mapping Kids;
  string topdir;

  [ topdir, Dirsize, Kids ] = input(argv[1..]);
  getdots(topdir, Dirsize, Kids);
  output(topdir, Dirsize, Kids, sizeof((string)sort(values(Dirsize))[-1]))
}
```

---

patternmatching.html

PLEAC-Pike

6. Pattern Matching

Introduction

Copying and Substituting Simultaneously

```
// @@INCOMPLETE@@


// by Scott McCoy [tag at cpan.org]


 /* Copying and Substituting Simultaniously */
dst = Regexp("this")->replace(src, "that");
 /* Copying and replacing...Same thing */
dst = Regexp("this")->replace(src, "that");
 /* Strip to basename */
 // Note this is best done with string manipulation functions, regexp here

 // wasteful.  Same is true in perl.

dst = Regexp("^.*/")->replace(argv[0], "");

string capword = Regexp("[a-z]+")->replace
        ("foo.bar",
            lambda (string c) {
                c[0] = upper_case(c[0]);
                return c;
            } );

/* For the captured substitution, I'll have to figure out PCRE. */
```

Matching Letters

Matching Words

Commenting Regular Expressions

Finding the Nth Occurrence of a Match

Matching Multiple Lines

Reading Records with a Pattern Separator

Extracting a Range of Lines

Matching Shell Globs as Regular Expressions

Speeding Up Interpolated Matches

Testing for a Valid Pattern

Program: tcgrep

Regular Expression Grabbag

---

fileaccess.html

PLEAC-Pike

---

7. File Access

Introduction

```
// if you are going to read the whole file, the most common way is to use

// Stdio.read_file()


string INPUT = Stdio.read_file("/usr/local/widgets/data");
if(!INPUT)
{
  werror("Couldn't open /usr/local/widgets/data for reading\n");
  exit(1);
}

foreach(INPUT/"\n";; string line)
{
  if(search(line, "blue")!=-1)
    write(line+"\n");
}

// if you need more control over the process you can get a filehandle with

// Stdio.File()
```

```
Stdio.File INPUT = Stdio.File("/usr/local/widgets/data", "r");
if(!INPUT)
{
  werror("Couldn't open /usr/local/widgets/data for reading\n");
  exit(1);
}

foreach(INPUT->line_iterator();; string line)
{
  if(search(line, "blue")!=-1)
    write(line+"\n");
}
INPUT->close();

//-----------------------------------------------------------


foreach(Stdio.stdin;; string line)            // reads from STDIN

{
  if(!sizeof(array_sscanf(line, "%*s%d")))
    werror("No digit found.\n");              // writes to STDERR

  write("Read: %s\n", line);                  // writes ot STDOUT

}
Stdio.stdout->close() || werror("couldn't close STDOUT\n") && exit(1);

// just as with Stdio.read_file(), there are convenience functions for wri

// Stdio.write_file() and Stdio.append_file()


Stdio.File logfile = Stdio.File("/tmp/log", "w");

// access modes are "r" for reading, "w" for writing and "a" for append

// default mode is "rw"


// to read a line you may use Stdio.File()->gets() or get a line_iterator(

// read lines from it.


object LOGFILE = logfile->line_iterator();
do
{
  string line=LOGFILE->value();
}
while(LOGFILE->next())
```

```
logfile->close();

// or use foreach as shown above.


// write() is actually a shortcut for Stdio.stdout->write()

// you could get yourself a different shortcut by assigning that to a vari


function write = logfile->write;     //  switch to LOGFILE for write();

write("Countdown initiated ...\n");
write = Stdio.stdout->write;         //  return to stdout

write("You have 30 seconds to reach minimum safety distance.\n");

// Stdio.File is unbuffered. a buffered version is provided by Stdio.FILE
```

Opening a File

```
// use Stdio.read_file(), Stdio.write_file() and Stdio.append_file() for

// convenience, or Stdio.File for precision and to get a filehandle.


string path;

// open file for reading

string file = Stdio.read_file(path);
Stdio.File file = Stdio.File(path, "r");

// open file for writing, create new file if needed, or else truncate old

Stdio.write_file(path, "content");
Stdio.File file = Stdio.File(path, "wc");

// same with setting access permissions

Stdio.write_file(path, "content", 0600);
Stdio.File file = Stdio.File(path, "wc", 0600);

// open file for writing, create new file, file must not exist

if(!file_stat(path))
  Stdio.write_file(path, "content");
Stdio.File file = Stdio.File(path, "wcx");
```

```
if(!file_stat(path))
  Stdio.write_file(path, "content", 0600);
Stdio.File file = Stdio.File(path, "wcx", 0600);

// open file for appending, create if necessary

Stdio.append_file(path, "content");
Stdio.File file = Stdio.File(path, "wac");

Stdio.append_file(path, "content", 0600);
Stdio.File file = Stdio.File(path, "wac", 0600);

// open file for appending, file must exist

Stdio.File file = Stdio.File(path, "wacx");

// open file for update, file must exist

string file = Stdio.read_file(path);
string updated = file+"foo"  // update contents of file

Stdio.write_file(path, updated);

Stdio.File file = Stdio.File(path);          // this is the default operat


// open file for update, file must not exist

Stdio.File file = Stdio.File(path, "rwcx");
```

Opening Files with Unusual Filenames

```
// since the filename is contained in a string, this problem does not appl
```

Expanding Tildes in Filenames

```
string filename;

if(filename[0] == "~")
{
  string user, path, home;
  [ user, path ] = array_sscanf(filename, "~%[^/]%s");
  if(user == "")
```

```
    home = getenv("HOME") || getenv("LOGDIR") || getpwuid(geteuid())[5];
  else
    home = getpwnam(user)[5];
  filename = home+path;
}
```

Making Perl Report Filenames in Errors

```
string path = "/tmp/fooo";
mixed error = catch
{
  Stdio.File file = Stdio.File(path, "r");
};

if(error)
{
  werror("Couldn't open %s for reading:\n", path);
  werror(error[0]);
}
// Couldn't open /tmp/fooo for reading:

// Failed to open "/tmp/fooo" mode "r" : No such file or directory
```

Creating Temporary Files

```
Stdio.File fh;
string name;
do
{
  name = "/tmp/"+MIME.encode_base64(random_string(10));
  fh = Stdio.File(name, "rwcx");
}
while(!fh)

atexit(lambda(){ fh->close(); rm(name); });


// if you don't really need the file to be on disk (or if /tmp is a ramdis

// but you need an object that behaves like a file, then use Stdio.FakeFil


fh = Stdio.FakeFile();
```

```
// and use fh like any other filehandle.
```

Storing Files Inside Your Program Text

```
// since the usual way to handle files is to read them into a string, ther
// assign your data to a string and work from there:

string data = "your data goes here";

// or for convenient multiline data:

string data = #"your data goes here
and here
and ends here";

// or use Stdio.FakeFile for a Stdio.File compatible interface
// see 7.5

//------------------------------------------------------------------

object stat = file_stat(__FILE__);
int raw_time = stat->ctime;
int size     = stat->size;
int kilosize = size/1024;

write("<P>Script size is %dk\n", kilosize);
write("<P>Last script update: %s\n", Calendar.Second(raw_time)->format_nic
```

Writing a Filter

Modifying a File in Place with Temporary File

Modifying a File in Place with -i Switch

Modifying a File in Place Without a Temporary File

Locking a File

Flushing Output

Reading from Many Filehandles Without Blocking

Doing Non-Blocking I/O

Determining the Number of Bytes to Read

Storing Filehandles in Variables

Caching Open Output Filehandles

Printing to Many Filehandles Simultaneously

Opening and Closing File Descriptors by Number

Copying Filehandles

Program: netlock

Program: lockarea

---

filecontents.html

---

8. File Contents

Introduction

Reading Lines with Continuation Characters

Counting Lines (or Paragraphs or Records) in a File

```
// Does not check file existence but return a correct value with empty (si
```

```
// Does count paragraphs correctly (does not count empty lines (\n\n))

int main(int argc, array(string) argv) {
    int count=0;
    object f = Stdio.FILE(argv[1]);
    foreach(f->line_iterator(f); int number; string paragraph) {
        count++;
        // write(number+" "+paragraph+"\n");

    }
    write("number of paragraphs= "+count+"\n");
    return 0;
}
```

Processing Every Word in a File

Reading a File Backwards by Line or Paragraph

Trailing a Growing File

Picking a Random Line from a File

Randomizing All Lines

Reading a Particular Line in a File

Processing Variable-Length Text Fields

Removing the Last Line of a File

Processing Binary Files

Using Random-Access I/O

Updating a Random-Access File

Reading a String from a Binary File

Reading Fixed-Length Records

Reading Configuration Files

Testing a File for Trustworthiness

Program: tailwtmp

Program: tctee

Program: laston

---

directories.html

PLEAC-Pike

---

9. Directories

Introduction

```
Stdio.Stat entry;

entry = file_stat("/bin/vi");
entry = file_stat("/usr/bin");
entry = file_stat(argv[1]);

// ------------
```

```
Stdio.Stat entry; int ctime, size;

entry = file_stat("/bin/vi");
ctime = entry->ctime;
size = entry->size;

// ------------
```

```
// A routine detecting whether a file is a 'text' file doesn't appear

// to exist, so have implemented the following [crude] function(s)

// which search for a LF / NEWLINE in the file:


// Usable with any file

int(0..1) containsText(Stdio.File file)
```

```
{
  string c;
  while ((c = file->read(1)) != NULL) { (c == NEWLINE) && return 1; }
  return 0;
}

// Alternate version, expects a buffered file [usually containing text]

int(0..1) containsText(Stdio.FILE file)
{
  int c;
  while ((c = file->getchar()) != EOF) { (c == LF) && return 1; }
  return 0;
}

// Yet another alternative - this time we cheat and use the *NIX 'file'

// utility :) !


int(0..1) isTextFile(string filename)
{
  return chop(Process.popen("file -bN " + filename), 1)  == "ASCII text";
}

// ----


containsText(Stdio.File(argv[1])) || write("File %s doesn't have any text

isTextFile(argv[1]) || write("File %s doesn't have any text in it\n", argv

// ------------


Filesystem.Traversion dirtree = Filesystem.Traversion("/usr/bin");

foreach(dirtree; string dir; string file)
{
  write("Inside %s is something called %s\n", chop(dir, 1), file);
}
```

Getting and Setting Timestamps

```
string filename = "example.txt";

Stdio.Stat fs = file_stat(filename);
int readtime = fs->atime, writetime = fs->mtime;
```

```
System.utime(filename, readtime, writetime);

// ----------------------------


constant SECONDS_PER_DAY = 60 * 60 * 24;

string filename = "example.txt";

Stdio.Stat fs = file_stat(filename);
int atime = fs->atime, mtime = fs->mtime;

atime -= 7 * SECONDS_PER_DAY; mtime -= 7 * SECONDS_PER_DAY;

System.utime(filename, atime, mtime);

// ----------------------------


argc != 1 || die("usage: " + argv[0] + " filename");

Stdio.Stat fs = file_stat(argv[1]);
int atime = fs->atime, mtime = fs->mtime;

Process.system(getenv("EDITOR") || "vi" + " " + argv[1]);

mixed result = catch { System.utime(argv[1], atime, mtime); };
(result == OK) || write("Error updating timestamp on file, %s!\n", argv[1
```

Deleting a File

```
string filename = "...";

rm(filename) || write("Can't delete, %s!\n", filename);

// ------------


int(0..1) rmAll(array(string) filelist)
{
  mixed result = catch
  {
    foreach(filelist, string filename) { rm(filename) || throw(PROBLEM); }
  };

  return result == OK;
}

// ----
```

```
array(string) filelist = ({"/tmp/x", "/tmp/y", "/tmp/z"});
rmAll(filelist) || write("Can't delete all files in array!\n");
// ----------------------------

void die(string msg, void|int(1..256) rc) { werror(msg + NEWLINE); exit(rc
// ----

string filename = "...";
rm(filename) || die("Can't delete " + filename);
// ----------------------------

array(string) filelist = ({"/tmp/x", "/tmp/y", "/tmp/z"});
int deleted, count = sizeof(filelist);
foreach(filelist, string filename) { rm(filename) && ++deleted; }
(deleted == count) || write("Could only delete %d of %d files\n", deleted,
```

Copying or Moving a File

```
string oldfile = "/tmp/old", newfile = "/tmp/new";
Stdio.cp(oldfile, newfile) || write("Error copying file\n");
// ----------------------------

string oldfile = "/tmp/old", newfile = "/tmp/new";
mixed result = catch { Stdio.write_file(newfile, Stdio.read_file(oldfile))
(result == OK) || write("Problem copying file %s to file %s\n", oldfile, n
// ----------------------------

// Note: This is a cross between, 'Process.system', which displays
```

```
// output on stdout, and, 'Process.popen', which does not display

// output [it returns it as a string] but does not return the status

// code

int system(string cmd)
{
  Stdio.File fout = Stdio.File(), ferr = Stdio.File();
  Stdio.File pout = fout->pipe(Stdio.PROP_IPC),
             perr = ferr->pipe(Stdio.PROP_IPC);

  int rc = Process.spawn(cmd, 0, pout, perr)->wait();

  pout->close(); destruct(pout); fout->close(); destruct(fout);
  perr->close(); destruct(perr); ferr->close(); destruct(ferr);

  return rc;
}

int(0..1) unixFileCopy(string oldfile, string newfile)
{
  string cmd = "cp --force --reply=yes " + oldfile + " " + newfile;
  return system(cmd) == OK;
}

int(0..1) vmsFileCopy(string oldfile, string newfile)
{
  string cmd = "copy " + oldfile + " " + newfile;
  return system(cmd) == OK;
}

// ----


string oldfile = "/tmp/old", newfile = "/tmp/new";

unixFileCopy(oldfile, newfile) || write("Problem copying file %s to file %

// ----------------------------


string oldfile = "/tmp/old", newfile = "/tmp/new";

mv(oldfile, newfile) || write("Problem moving / renaming file %s to file %
```

Recognizing Two Names for the Same File

```
mapping(array(int):int) seen = ([]);

// ----


void do_my_thing(string filename)
{
  Stdio.Stat fs = file_stat(filename);
  array(int) arr = aggregate(fs->inode, fs->dev);

  // Could do this [apply a lambda assigned to variable 'p']:

  //

  //    ... || (p(arr), seen[arr] = 1);

  //

  //    function p = lambda(array(int) arr) { ... };

  //

  // to process a file that has not previously been seen


  (seen[arr] && (seen[arr] += 1)) || (seen[arr] = 1);
}

// ----------------------------


constant SEP = ":"; mapping(array(int):string) seen = ([]);

// ----


array(string) files = ({"f1.txt", "f2.txt", "f3.txt"});

foreach(files, string filename)
{
  Stdio.Stat fs = file_stat(filename);
  array(int) arr = aggregate(fs->inode, fs->dev);
  (seen[arr] && (seen[arr] += (SEP + filename))) || (seen[arr] = filename)
}

// ----
```

```
array(array(int)) idxarr = indices(seen); sort(idxarr);

foreach(idxarr, array(int) inodev)
{
  foreach(seen[inodev] / SEP, string filename)
  {
    // ... do stuff with each filename ...

    write("%s\n", filename);
  }
}
```

Processing All Files in a Directory

```
string dirname = "..."; array(string) DIR = get_dir(dirname);

foreach(DIR, string filename)
{
  string path = dirname + "/" + filename;
  // ... do something with 'path' ...

}

// ----------------------------


string dirname = "/usr/local/bin"; int|array(string) DIR = get_dir(dirname

DIR || die("Can't open " + dirname);

write("Text files in %s are:\n", dirname);

foreach(DIR, string filename)
{
  string path = dirname + "/" + filename;

  // 'isTextFile' defined in an earlier section

  isTextFile(path) && write("%s\n", filename);
}

// ----------------------------


// '.' and '..' don't show up in a 'get_dir'-generated array


// ----------------------------
```

```
array(string) plain_files(string dirname)
{
  // 'filter' procedure

  function fp =
    lambda(string filename, string dirname)
    {
      // 'isTextFile' defined in an earlier section

      return !has_prefix(filename, ".") && isTextFile(dirname + "/" + file
    };

  // 'map' procedure

  function mp =
    lambda(string filename, string dirname)
    {
      return dirname + "/" + filename;
    };

  array(string) paths = map(filter(get_dir(dirname), fp, dirname), mp, dir

  sort(paths);

  return paths;
}
```

Globbing, or Getting a List of Filenames Matching a Pattern

```
// A 'glob' workalike that filters using regular expressions

//

// Note: Pike offers many non-regexp-based string pattern matching

// functions [e.g. 'has_prefix' and other 'has_...' functions,

// 'search', etc]. These are preferable in many situations as they are

// much faster than regexprs. However, code shown here mostly uses

// regexprs in order to better match the Perl examples

//

int(0..1)|array(string) grep(string regexp, string|array(string) arr)
```

```
{
  if (stringp(arr)) return Regexp.match(regexp, arr);

  if (arrayp(arr))
  {
    function fp =
      lambda(string filename, string regexp)
      {
        return Regexp.match(regexp, filename);
      };

    return filter(arr, fp, regexp);
  }

  return 0;
}

// ----------------------------


string dirname = "...";
int|array(string) filenames = glob("*.c", get_dir(dirname));

// ------------


string dirname = "...";
int|array(string) filenames = grep("\.c$", get_dir(dirname));

// ------------


string dirname = "...";
int|array(string) filenames = grep("\.[CHch]$", get_dir(dirname));

// ----------------------------


string dirname = "...";
int|array(string) dir = get_dir(dirname);

dir || die("Couldn't open " + dirname + " for reading");

//

// Note: Pike arrays are immutable, so we use a mapping to emulate

// mutable arrays by using a numeric index as the key :)

//

mapping(int:string) files = ([]); int idx = -1; string path;
```

```
foreach(dir, string file)
{
  if (!grep("\.[CHch]$", file)) continue;
  path = dirname + "/" + file;
  isTextFile(path) && (files[++idx] = path);
}

//

// Note: Traverse a mapping-based, emulated array in index order:

//

//  foreach(sort(indices(files)), int i)

//  {

//    write("%d -> %s\n", i, files[i]);

//  }

//
```

Processing All Files in a Directory Recursively

```
//

// Routine inspired by library function, 'Stdio.recursive_rm'. A little

// extra code helped make it more generally useful

//

void|mixed process_directory(string path, function(string, mixed ... : voi
{
  Stdio.Stat file = file_stat(path, 1); if (!file) return 0;

  if (file->isdir)
    if (array(string) files = get_dir(path))
      foreach(files, string file)
        process_directory(path + "/" + file, op, @extra_args);

  return op(path, @extra_args);
}

// ----------------------------
```

```
array(string) dirlist = ({ "/tmp/d1", "/tmp/d2", "/tmp/d3" });

// Do something with each directory in the list

foreach(dirlist, string dir)
{
  // Delete directory [if empty]     -> rm(dir);

  // Make it the 'current directory' -> cd(dir);

  // Get list of files it contains   -> array(string) filelist = get_dir(d

  // Get directory metadata          -> Stdio.Stat ds = file_stat(dir);

}

// ------------
```

```
array(string) dirlist = ({ "/tmp/d1", "/tmp/d2", "/tmp/d3" });

function pf =
  lambda(string path)
  {
    // ... do something to the file or directory ...

    write("%s\n", path);
  };

// For each directory in the list ...

foreach(dirlist, string dir)
{
  int|array(string) filelist = get_dir(dir);

  if (!filelist) { write("%s does not exist\n", dir); continue; }
  if (sizeof(filelist) == 0) { write("%s is empty\n", dir); continue; }

  // For each file / directory in the directory ...

  foreach(filelist, string filename)
  {
    // Apply function to process the file / directory

    pf(dir + "/" + filename);
  }
}

// ------------
```

```
// Special steps need to be taken in above routines to distinguish

// between files and directories. Easiest to abstract out directory

// traversal into a single routine [so allowing for recursive traversal

// of entire tree], and have it apply a lambda to each file


array(string) dirlist = ({ "/tmp/d1", "/tmp/d2", "/tmp/d3" });

function pf =
  lambda(string path)
  {
    // ... do something to the file or directory ...

    write("%s\n", path);
  };

// For each directory in the list ...

foreach(dirlist, string dir)
{
  process_directory(dir, pf);
}

// ----------------------------


void accum_filesize(string path, array(int) accum)
{
  int|Stdio.Stat fs = file_stat(path);

  // Accumulate size only if it is a regular file

  (fs && fs->isreg) && (accum[0] += fs->size);
}

// ------------


// Verify arguments ...

argc == 2 || die("usage: " + argv[0] + " dir");
Stdio.Stat fs; string dir = argv[1];
((fs = file_stat(dir)) && fs->isdir) || die(dir + " does not exist / not a

// Collect data [use an array to accumulate results]

array(int) dirsize = ({0});
process_directory(dir, accum_filesize, dirsize);
```

```
// Report results

write("%s contains %d bytes\n", dir, dirsize[0]);

// ----------------------------


void biggest_file(string path, array(mixed) biggest)
{
  int|Stdio.Stat fs = file_stat(path);

  if (fs && fs->isreg && biggest[1] < fs->size)
  {
    biggest[0] = path; biggest[1] = fs->size;
  }
}

// ------------


// Verify arguments ...

argc == 2 || die("usage: " + argv[0] + " dir");
Stdio.Stat fs; string dir = argv[1];
((fs = file_stat(dir)) && fs->isdir) || die(dir + " does not exist / not a

// Collect data [use an array to store results]

array(mixed) biggest = ({"", 0});
process_directory(dir, biggest_file, biggest);

// Report results

write("Biggest file is %s containing %d bytes\n", biggest[0], biggest[1]);

// ----------------------------


void youngest_file(string path, array(mixed) youngest)
{
  int|Stdio.Stat fs = file_stat(path);

  if (fs && fs->isreg && youngest[1] > fs->ctime)
  {
    youngest[0] = path; youngest[1] = fs->ctime;
  }
}

// ------------


// Verify arguments ...
```

```
argc == 2 || die("usage: " + argv[0] + " dir");
Stdio.Stat fs; string dir = argv[1];
((fs = file_stat(dir)) && fs->isdir) || die(dir + " does not exist / not a

// Collect data [use an array to store results]

array(mixed) youngest = ({"", Int.NATIVE_MAX});
process_directory(dir, youngest_file, youngest);

// Report results

write("Youngest file is %s dating %s\n", youngest[0], ctime(youngest[1]));

// ----------------------------


void print_name_if_dir(string path)
{
  int|Stdio.Stat fs = file_stat(path);
  if (fs && fs->isdir) write("%s\n", path);
}

// ------------


// Verify arguments ...

argc == 2 || die("usage: " + argv[0] + " dir");
Stdio.Stat fs; string dir = argv[1];
((fs = file_stat(dir)) && fs->isdir) || die(dir + " does not exist / not a

// Print directory names

process_directory(dir, print_name_if_dir);
```

Removing a Directory and Its Contents

```
// Easy way - recommended

int(0..1) rmTree(string dirname) { return Stdio.recursive_rm(dirname); }

// ----


string dirtree = "/tmp/dirtree";

rmTree(dirtree) || write("Problem removing directory tree %s\n", dirtree);
```

```
// ----------------------------

// Another way, but unnecessary - probably for customised deletions only

int(0..1) rmTree(string dirname) { return process_directory(dirname, rm);

// ----

string dirtree = "/tmp/dirtree";

rmTree(dirtree) || write("Problem removing directory tree %s\n", dirtree);
```

Renaming Files

```
// A list of file names

array(string) names = ({"f1.txt", "f2.txt", "f3.txt"});

// Dynamically assigned 'rename' procedure - can be reassigned at any time

function rename = lambda(string name) { return replace(name, ".txt", ".tex

// Process all files

foreach(names, string name)
{
  // Generate new name from existing name by applying 'rename' procedure

  string newname = rename(name);

  // Perform actual rename task on file

  mv(name, newname) || write("Could not rename %s to %s\n", name, newname)
}

// ----------------------------

// Slightly different to the Perl example, though it does use regexp

// and intent is roughly the same.

//

// pike SCRIPTNAME '\.txt$' '.text' f1.txt f2.txt df3.txg

//
```

```
//    f1.txt  -> f1.text

//    f2.txt  -> f2.text

//    df3.txg -> df3.txg [no change]

//


argc > 2 || die("usage: " + argv[0] + " expr repl files...");

string expr = argv[1], repl = argv[2];

foreach(argv[3..], string name)
{
  string newname = Regexp.replace(expr, name, repl);

  if (!equal(name, newname))
    mv(name, newname) || write("Could not rename %s to %s\n", name, newnam
}
```

Splitting a Filename into Its Component Parts

```
string file_extension(string filename, void|string separator)
{
  return (filename / (separator || "."))[-1];
}

mapping(string:string) file_parse(string path)
{
  return
    mkmapping(({"dirname", "basename", "extension"}),
      ({dirname(path), basename(path), file_extension(basename(path))}));
}

// ----------------------------


string path = "/tmp/dirtree/s/s1/s1.txt";

// ----


string dir = dirname(path);
string base = basename(path);

mapping(string:string) pm = file_parse(path);
write("%s\n", pm["dirname"]);
```

```
write("%s\n", pm["basename"]);
write("%s\n", pm["extension"]);

// ------------


string path = "/usr/lib/libc.a";

// ----


string dir = dirname(path);
string base = basename(path);

write("dir is %s, file is %s\n", dir, base);

// ------------


string path = "/usr/lib/libc.a";

// ----


mapping(string:string) pm = file_parse(path);

write("dir is %s, name is %s, extension is %s\n",
  pm["dirname"], pm["basename"], "." + pm["extension"]);

// ----------------------------


// Handle as a general purpose parse task

string path = "Hard%20Drive:System%20Folder:README.txt";

// ----


mapping(string:string)
  pm = mkmapping(({"drive", "folder", "filename"}),
                 replace(path, "%20", " ") / ":"),

  fm = mkmapping(({"name", "extension"}),
                 pm["filename"] / ".");

write("dir is %s, name is %s, extension is %s\n",
  pm["drive"] + ":" + pm["folder"],
  fm["name"], "." + fm["extension"]);

// ----------------------------
```

```
// See implementation for 'file_extension' function above
```

Program: symirror

```
@@INCOMPLETE@@
@@INCOMPLETE@@
```

Program: lst

```
@@INCOMPLETE@@
@@INCOMPLETE@@
```

---

subroutines.html

PLEAC-Pike

---

10. Subroutines

Introduction

```
// Here, in this simple example, 'greeted', is used as a 'global'
// variable. In a more complex program, however, this would not be
// the case [subsequent sections exlain why]
int greeted;
// ----

void hello()
```

```
{
  write("hi there!, this procedure has been called %d times\n", ++greeted)
}

int how_many_greetings()
{
  return greeted;
}

// ------------


int main()
{
  hello();
  int greetings = how_many_greetings();
  write("bye there!, there have been %d greetings so far\n", greetings);
}

// ----------------------------


// Alternate means of defining functions [could, optionally, have also

// included type information in 'function' declaration]; could also

// have been done within scope of 'main'


int greeted;

// ----


function hello = lambda()
  {
    write("hi there!, this procedure has been called %d times\n", ++greete
  };

function how_many_greetings = lambda() { return greeted; };

// ------------


int main()
{
  hello();
  int greetings = how_many_greetings();
  write("bye there!, there have been %d greetings so far\n", greetings);
}
```

Accessing Subroutine Arguments

```
// Subroutine parameters are named, that is, access to these items from
// within a function is reliant on their being named in the parameter
// list [together with mandatory type information], something which is
// in line with many other commonly-used languages

float hypotenuse(float side1, float side2)
{
  // Arguments passed to this function are accessable as, 'side1',
  // and 'side2', respectively, and each is expected to be a 'float'
  // type
  return side1 * side1 + side2 * side2;
}
// ----

// 'side1' -> 3.0
// 'side2' -> 4.0
float diag = hypotenuse(3.0, 4.0);
// ------------

// However, Pike also allows parameters [and return types where applicable
// * To have one of a set of types [see (1)]
// * To have a generic type [see (2)]
// * To be optional, in which case any arguments are packaged as an
//   array, and array notation needed to access each item [see (3)]

// (1). Here the function will accept either 'int' or 'float'
// arguments, and perform runtime type checking to identify what is
// supplied
```

```
float hypotenuse(int|float side1, int|float side2)
{
  // If 'int' arguments passed. convert to 'float'

  float s1 = intp(side1) ? (float) side1 : side1;
  float s2 = intp(side2) ? (float) side2 : side2;

  return s1 * s1 + s2 * s2;
}

// ----


// Both are legal calls

float diag = hypotenuse(3.0, 4.0);
float diag = hypotenuse(3, 4);

// ------------


// (2). Here the function still expects to be called with two arguments

// but each may be of *any* type [admittedly a very contrived example

// of little utility except for illustrative value]. Such a function

// is almost entirely reliant on careful runtime type checking if it

// is to behave reliably

float hypotenuse(mixed side1, mixed side2)
{
  if (stringp(side1)) { ... }
  if (arrayp(side1)) { ... }
  if (objectp(side1)) { ... }
  ...
}

// ----


// All are legal calls

float diag = hypotenuse(3.0, 4.0);
float diag = hypotenuse(3, 4);
float diag = hypotenuse("3", "4");
float diag = hypotenuse(({3}), ({4}));

// ------------
```

```
// (3). Here, the function is defined to accept two, mandatory
// parameters [still accessable via name], then a set of zero or more
// optional parameters, which are accessable within the function body
// via an array [the placeholder, 'args', represents an array of zero
// or more elements each corresponding to one of the passed arguments
float hypotenuse(float side1, mixed side2, mixed ... args)
{
  // Mandatory parameters still accessable as usual
  ... side1 ... side2 ...
  // Total number of arguments passed to function determinable via:
  int total_passed_args = query_num_arg();
  // 'args' contains all optional arguments: 0 - N
  int optional_args = sizeof(args);
  // Process variable arguments ...
  foreach(args, mixed arg)
  {
    ... if (stringp(arg)) { ... }
  }
  ...
}
// ----
// All are legal calls
float diag = hypotenuse(3.0, 4.0);
float diag = hypotenuse(3.0, 4.0, "a");
float diag = hypotenuse(3.0, 4.0, lambda(){ return 5; }, "fff");
float diag = hypotenuse(3.0, 4.0, 1, "x", ({ 6, 7, 9 }));
// ----------------------------
// Modifies copy
array(int|float) int_all(array(int|float) arr)
{
```

```
  array(int|float) retarr = copy_value(arr);
  int i; for(int i; i < sizeof(retarr); ++i) { retarr[i] = (int) arr[i]; }
  return retarr;
}

// Modifies original

array(int|float) trunc_all(array(int|float) arr)
{
  int i; for(int i; i < sizeof(arr); ++i) { arr[i] = (int) arr[i]; }
  return arr;
}

// ----


array(int|float) nums = ({1.4, 3.5, 6.7});

// Copy modified - 'ints' and 'nums' separate arrays

array(int|float) ints = int_all(nums);
write("%O\n", nums);
write("%O\n", ints);

// Original modified - 'ints' acts as alias for 'nums'

ints = trunc_all(nums);
write("%O\n", nums);
write("%O\n", ints);
```

Making Variables Private to a Function

```
void some_func()
{
  // Variables declared within a function are local to that function

  mixed variable = something;
}

// ----------------------------


// Assuming these are defined at file level, that is, outside of 'main'

// or any other function they are accessable by every other member of

// the same file [and if this file (read: class or program) is the

// only one comprising the 'system', they are effectively 'global']
```

```
string name = argv[1]; int age = (int) argv[2];

int c = fetch_time();

int condition;

// ------------


int run_check()
{
  ...
  condition = 1;
}

int check_x(int x)
{
  string y = "whatever";

  // Whilst 'run_check' has access to 'name', 'age', and 'c' [because

  // these are declared at a higher scope], it does not have access to

  // 'y' or any other locally defined variable

  run_check();

  // 'run_check' will have updated 'condition'

  if (condition) write("got x: %d\n", x);
}
```

Creating Persistent Private Variables

```
// Pike does not implement C style 'static' variables [i.e. persisent

// local variables], nor does it implement C++ style 'class variables'

// [oddly enough, also implemented in C++ via use of the 'static'

// keyword], both of which could be used to implement solutions to the

// problems presented in this section. Also, there is no direct

// equivalent to Perl's 'BEGIN' block [closest equivalent is the

// class 'create' method]. So, to solve a problem like implementing a
```

```
// 'counter':

//

// * Use Pike's OOP facilities [simple, natural]

// * Use closures [somewhat unwieldly, but possible]


// OOP Approach

class Counter
{
  private int counter;

  static void create(int start) { counter = start; }
  public int next() { return ++counter; }
  public int prev() { return --counter; }
}

// ----


int main()
{
  Counter counter = Counter(42);

  write("%d\n", counter->next());
  write("%d\n", counter->prev());
}

// ----------------------------


// A refinement of the previous implementation that mimics 'static'

// variables


class Static
{
  // 'static' variable that is shared by all instance of 'Counter'

  int counter;

  class Counter
  {
    public int next() { return ++counter; }
    public int prev() { return --counter; }
  }

  Counter make() { return Counter(); }
```

```
  public void create(int counter_) { counter = counter_; }
}

// ----


int main()
{
  Static mkst = Static(42);

  Static.Counter counter_1 = mkst->make();
  Static.Counter counter_2 = mkst->make();

  // Same value of, 'counter', is accessed by each object

  write("%d\n", counter_1->next());
  write("%d\n", counter_1->next());

  write("%d\n", counter_2->next());
  write("%d\n", counter_2->prev());
}

// ----------------------------


// Closure Approach [Admittedly somewhat contrived: a Scheme overdose ;) !


function(string : function(void : int)) make_counter(int start)
{
  int counter = start;
  int next_counter() { return ++counter; };
  int prev_counter() { return --counter; };

  return
    lambda(string op)
    {
      if (op == "next") return next_counter;
      if (op == "prev") return prev_counter;
      return 0;
    };
}

int next_counter(function(string : function(void : int)) counter)
{
  return counter("next")();
}

int prev_counter(function(string : function(void : int)) counter)
{
  return counter("prev")();
```

```
}

// ----


int main()
{
  function(string : function(void : int)) counter = make_counter(42);

  write("%d\n", next_counter(counter));
  write("%d\n", prev_counter(counter));
}
```

Determining Current Function Name

```
// It's possible to obtain a great deal of program metadata through the

// following sets of library functions:

//

// * 'this_object', and the sets of 'object_...' and 'program_...'

//   functions

// * 'Program' module [provides object inheritance metadata]

//

// The use of, 'this_object', in particular, allows the current object

// instance to be interrogated like a hash table i.e. all variables and

// methods are accessable as hash table entries.

//

// Unfortunately, however, it doesn't appear possible to obtain the

// current method / function name, at least, not without resorting

// to tricks like embedding a string in each method explicitly naming

// it.

//

// An example of program metadata use appears in chapter 'Objects and
```

```
// Ties'. Since the function name cannot, AFAICT, be obtained, the
// current section is not implemented.
//
```

Passing Arrays and Hashes by Reference

```
// Procedure parameters are passed by reference [read: the handle or
// address (or whatever) of an object is passed and is used to uniquely
// identify that object], so there is no special treatment required.
// If an argument represents a mutable object then care should be taken
// to not mutate the object within the function, either by making a copy
// of the object [e.g. use 'copy_value' to clone it], or by using a
// 'read-only' control structure like 'foreach' [if applicable] to
// access it

int|array(int) array_diff(array(int) a, array(int) b)
{
  // Ensure an array copy is made ...

  int|array(int) ret = sizeof(a) != sizeof(b) ? 0 : copy_value(a);
  if (!ret) ret;

  // ... transformed, and returned

  for (int i; i < sizeof(ret); ++i) { ret[i] -= b[i]; }; return ret;
}

// ----------------------------

int|array(int) add_vec_pair(array(int) a, array(int) b)
{
  int vecsize = sizeof(a);

  // Ensure an array copy is made ...

  int|array(int) ret = vecsize != sizeof(b) ? 0 : allocate(vecsize);
  if (!ret) ret;
```

```
  // ... transformed, and returned

  for (int i; i < vecsize; ++i) { ret[i] = a[i] + b[i]; }; return ret;
}

// ----


array(int) a = ({1, 2}), b = ({5, 8});
array(int) c = add_vec_pair(a, b);
write("%O\n", c);
```

Detecting Return Context

```
// Just as for subroutine parameters, Pike allows variation in return

// types, where it may be of a specific type, one of a set of types, or

// a generic return type. Whilst the Perl examples require that the

// user to nominate a return type when the function is called, in Pike

// it is handled in one of two ways:

//

// * Ensure function and receiving variable type match [see (1)]

// * Use generic receiving variable, and type check [see (2)]


// (1). Subroutine has set of return types. Whilst type checking

// does occur [thus user code only need handle these known types

// because other types won't be allowed], caller/ receiver needs to

// type check so as correctly handle known cases

int|array(int)|string mysub()
{
  ...
  return 5;
  ...
  return ({5});
  ...
  return "5";
}
```

```
// ----


int|array(int)|string receiver = mysub();

if (intp(receiver)) { ... }
if (arrayp(receiver)) { ... }
if (stringp(receiver)) { ... }
...

// ----------------------------


// (2). Subroutine has generic return type, so no type checking occurs.

// It is up to the caller / receiver to thoroughly type check lest

// some unforseen type be returned and possibly mishandled

mixed mysub()
{
  ...
  return 5;
  ...
  return ({5});
  ...
  return "5";
}

// ----


mixed receiver = mysub();

if (intp(receiver)) { ... }
if (arrayp(receiver)) { ... }
if (stringp(receiver)) { ... }
...
```

Passing by Named Parameter

```
// Pike doesn't directly support named / keyword parameters, but these

// can be easily mimiced using an array of mappings as function arguments

// The mappings, themselves, could be implemented via a custom class

// [see (1)], or via the 'mapping'type [see (2) - Perl examples]
```

```
// (1)

class KeyedValue
{
  string key; mixed value;

  static void create(string key_, mixed value_)
  {
    key = key_; value = value_;
  }
}

void the_func(array(KeyedValue) keyargs)
{
  foreach(keyargs, KeyedValue kv)
  {
    write("Key: %10s|Value: %10O\n", kv->key, kv->value);
  }
}

// ----


int main()
{
  array(KeyedValue) keyargs =
    aggregate(KeyedValue("name", "Bob"),
              KeyedValue("age", 36),
              KeyedValue("income", 51000));

  the_func(keyargs);
}

// ----------------------------


// (2)

class RaceTime
{
  int time; string dim;

  static void create(int time_, string dim_)
  {
    time = time_; dim = dim_;
  }
}

void the_func(mapping(string : RaceTime) ... args)
{
```

```
  int start_time, finish_time, increment_time;
  string start_dim, finish_dim, increment_dim;

  foreach(args, mapping(string : RaceTime) arg)
  {
    arg["start"] && (start_time = arg["start"]->time, start_dim = arg["sta
    arg["finish"] && (finish_time = arg["finish"]->time, finish_dim = arg[
    arg["increment"] && (increment_time = arg["increment"]->time, incremen
  }

  write("times: start %d, finish %d, increment %d\n",
        start_time, finish_time, increment_time);
}

// ----


int main()
{
  array(mapping(string : RaceTime)) named_args =
    ({
      (["increment" : RaceTime(20, "s")]),
      (["start" : RaceTime(5, "m")]),
      (["finish" : RaceTime(3, "m")]) });

  // Package arguments as array for passing to function

  the_func(@named_args);

  named_args =
    ({
      (["start" : RaceTime(5, "m")]),
      (["finish" : RaceTime(30, "m")]) });

  // Ditto

  the_func(@named_args);

  // Pass argument(s) directly in argument list

  the_func((["finish" : RaceTime(30, "m")]));
}
```

Skipping Selected Return Values

```
// It is languages that support pattern matching, such as Prolog, Oz and

// SML, that tend to offer such facilities. These languages all offer
```

```
// a 'match all and throw away' operator that can be used in place of
// an identifier name(s), and have the resultant value(s) be discarded.
// Such a facility helps keep code uncluttered because only values
// that are required need to be named.
//
// Pike does not implement pattern matching, so does not sport such an
// operator, nor any equivalent facility. Thus, none of the examples in
// this section are directly implementable.
//
```

Returning More Than One Array or Hash

```
// Pike supports the return of a single value from a function. Where
// multiple values need to be returned, they can be packaged as an
// aggregate such as an array or mapping, and *that* item returned.
// The caller / receiver would, of course, be responsible for
// appropriately extracting required elements from that returned item
// (this process could be hardcoded, or generalised using extensive
// runtime type checking). Alternatively, a custom class encapsulating
// the return values can be used and an instance of that item returned
// and processed
//

array(mixed) somefunc()
{
  array(int) arr = ({1, 2, 3});
  mapping(string : int) hash = (["x" : 1, "y" : 2, "z" : 3]);

  // Return an array containing an array and a hash
```

```
  return aggregate(arr, hash);
}

// ----

// Get return array

array(mixed) arr = somefunc();

// Extract and process elements

foreach(arr, mixed item)
{
  write("Return item has type: %t, value: %O\n", item, item);
}

// ----------------------------


class RetValues
{
  array(int) arr; mapping(string : int) hash;

  static void create()
  {
    arr = aggregate(1, 2, 3);
    hash = aggregate_mapping("x", 1, "y", 2, "z", 3);
  }
}

RetValues somefunc() { return RetValues(); }

// ----


RetValues rv = RetValues();

write("Return item has type: %t, value: %O\n", rv->arr, rv->arr);
write("Return item has type: %t, value: %O\n", rv->hash, rv->hash);
```

Returning Failure

```
// Pike offers a very simple, consistent means of 'returning failure':

// return 0 [representing 'false'] when a task does not succeed, otherwise

// return whatever was the expected value. This design is extensively
```

```
// used in the Pike library, and is well supported by the language in
// that:
// * Alternate return types may be specified
// * A 'mixed' return type, indicating possible return of any type
//   value, may be used

void die(string msg, void|int(1..256) rc) { werror(msg + NEWLINE); exit(rc

// ----

int|array(string) afunc()
{
  ...
  if (ok)
    // ... return an array ...
  else
    // failure, so return 0
    return 0;
}

// ----

int main()
{
  int|array(string) arr = afunc();
  if (!arr) die("Error with 'afunc' ...");
  // ok, so use 'arr' ...
}
```

Prototyping Functions

```
// Whether Pike is seen to support prototyping depends on the definition
// of this term used:
//
```

```
// * Prototyping along the lines used in Ada, Modula X, and even C / C++,
//   in which a procedure's interface is declared separately from its
//   implementation, is *not* supported
//
// * Prototyping in which, as part of the procedure definition, parameter
//   information must be supplied. This is a requirement in Pike in that
//   parameter number, names and type, must be given. Return types must
//   also be specified, but there is an exeption when using lambdas

void func_with_no_arg() { ... }
void func_with_one_arg(int arg1) { ... }
void func_with_two_arg(int arg1, string arg2) { ... }
void func_with_three_arg(int arg1, string arg2, float arg3) { ...}
// ----

// Return type, 'void', specified
function f = lambda(int arg1 : void) { ... }
// Return type not specified, defaults to 'mixed'
function g = lambda(int arg1) { ... }
```

Handling Exceptions

```
// Like so many modern languages, Pike implements exception handling
// using the 'catch' and 'throw' keywords

// ----

// Not exactly like the Perl example, but a way of immediately exiting
```

```
// from an application [note: using, 'exit', prevents any of the 'atexit'

// call backs from executing, so application cleanup may be compromised]

void die(string msg, void|int(1..256) rc) { werror(msg + NEWLINE); exit(rc

// ----

die("some message");

// ----------------------------

int(0..1) rmAll(array(string) filelist)
{
  // 'result' will be 0 if the 'catch' block succeeds

  mixed result = catch
  {
    foreach(filelist, string filename) { rm(filename) || throw(PROBLEM); }
  };

  // Return value of 'catch' block can be tested, and appropriate

  // action taken; here, a non-zero return code will be returned

  // [like many library functions] to indicate failure

  return result == OK;
}

// ----

// Attempt to remove the following files ...

array(string) files = ({"...", "...", "..."});

rmAll(files) || die("Could not remove all files - exiting");
```

Saving Global Values

```
// Global variable

int age = 18;
```

```
// ----

void print_age()
{
  // Global value, 'age', is accessed

  write("Age is %d\n", age);
}

// ------------

int main()
{
  // A local variable named, 'age' will act to 'shadow' the globally

  // defined version, thus any changes to, 'age', will not affect

  // the global version

  int age = 5;

  // Prints 18, the current value of the global version

  print_age();

  // Local version is altered, *not* global version

  age = 23;

  // Prints 18, the current value of the global version

  print_age();
}

// ----------------------------

// Global variable

int age = 18;

// ----

void print_age()
{
  // Global value, 'age', is accessed

  write("Age is %d\n", age);
```

```
}

// ------------

int main()
{
  // Here no local version declared: any changes affect global version

  age = 5;

  // Prints 5, the new value of the global version

  print_age();

  // Global version again altered

  age = 23;

  // Prints 23, the new value of the global version

  print_age();
}

// ----------------------------

// Global variable

int age = 18;

// ----

void print_age()
{
  // Global value, 'age', is accessed

  write("Age is %d\n", age);
}

// ------------

int main()
{
  // Global version value saved into local version

  int age = global::age;

  // Prints 18, the new value of the global version
```

```
  print_age();

  // Global version this time altered

  global::age = 23;

  // Prints 23, the new value of the global version

  print_age();

  // Global version value restored from saved local version

  global::age = age;

  // Prints 18, the restored value of the global version

  print_age();
}
```

Redefining a Function

```
// Define functions - will be considered constant / unchangeable

void grow() { write("grow\n"); }
void shrink() { write("shrink\n"); }

// ----


// Execute functions: 'grow', 'shrink' output respectively

grow(); shrink();

// Attempt to redefine, 'grow' fails because it is considered a

// constant value:

//

//   grow = shrink;

//


// However, it is possible to bind 'shrink' to a new local variable

// called, 'grow'

function grow = shrink;
```

```
// Execute functions: 'shrink', 'shrink' output respectively because

// local 'grow' shadows global version, and it is referencing the

// code for 'shrink'

grow(); shrink();

// ------------

// Define functions by assigning lambdas to global variables

function(void : void) grow = lambda() { write("grow\n"); }
function(void : void) shrink = lambda() { write("shrink\n"); }

// ----

// Execute functions: 'grow', 'shrink' output respectively

grow(); shrink();

// Attempt to redefine, 'grow' successful since a simple variable

// assignment is being performed

grow = shrink;

// Execute functions: 'shrink', 'shrink' output respectively - 'grow'

// has, effectively, been 'redefined' [note: reference to original

// 'grow' code has been lost (but it could have been saved, then

// restored)]

grow(); shrink();

// ---------------------------

function barney = lambda() { ... };

// ...

// 'fred' is now an alias for the code attached to 'barney'

function fred = barney;
```

```
// ----------------------------


function red = lambda(string text)
  {
    return "<FONT COLOR='red'>" + text + "</FONT>";
  };

// ----


write("%s\n", red("careful here"));

// ------------


function colour_font = lambda(string colour, string text)
  {
    return "<FONT COLOR='" + colour + "'>" + text + "</FONT>";
  };

function red = lambda(string text) { return colour_font("red", text); };
function green = lambda(string text) { return colour_font("green", text);

// ... more 'colour' functions ...


// ----


write("%s\n", red("careful here"));
write("%s\n", green("careful there"));
// ...


// ------------


// Pike offers the 'compile' family of functions that allow for the

// runtime compilation and [in combinaton with other Pike functions]

// the subsequent execution, of Pike code, obtained either as a

// dynamically-generated string, or loaded from file / URL. This,

// AFAICT, is the Pike feature closest to that of the 'eval' function

// found in languages like Scheme. The example here is rather

// contrived and unwieldly, but it does show how code is generated,
```

```
// compiled, and executed, and it *does* closely follow the Perl code

//


// Assemble text needed to build function

string build_colour_func(string colour)
{
  // Could also use library function, 'sprintf', to build string

  string bodytext = "\"<FONT COLOR='" + colour + "'>\" + text + \"</FONT>\
  return "string " + colour + "(string text) { return " + bodytext + "; };
}

int main(int argc, array(string) argv)
{
  // 1. Generate source code. A function is built for each colour by

  //    calling, 'build_colour_func', and all the text collected into

  //    'cf_text'

  array(string) colours =
    ({"red", "blue", "green", "yellow", "orange", "purple", "violet"});

  string cf_text = "";

  foreach(colours, string colour)
  {
    cf_text += build_colour_func(colour);
  }

  // 2. Compile generated source code, and make it accessable to the

  //    current program. These two steps are, here, combined for brevity,

  //    but consist of the following:

  //

  //        program prog = compile_string(cf_text);

  //        object cf_code = prog();

  //

  //    The latter step sees the code's 'create' method called for

  //    initialisation, and also makes items accessable via:

  //
```

```
  //        cf_code[ITEMNAME]
  //
  //    For example, the function, 'red', may be:
  //        referenced -> cf_code["red"]
  //        applied    -> cf_code["red"](" ... ")
  //
  object cf_code = compile_string(cf_text)();
  // 3. Apply the generated functions
  mapping(string:string) colours_and_text =
    (["red":"baron", "blue":"zephyr", "green":"beret",
      "yellow":"ribbon", "orange":"county", "purple":"haze",
      "violet":"temper"]);
  foreach(indices(colours_and_text), string colour)
  {
    // Get relevant function
    function colour_func = cf_code[colour];
    // Apply function with relevant argument(s)
    write("%s\n", colour_func(colours_and_text[colour]));
    // Or, above lines can be replaced with:
    //     write("%s\n", cf_code[colour](colours_and_text[colour]));
  }
}
```

Trapping Undefined Function Calls with AUTOLOAD

```
// @@INCOMPLETE@@

// in your class
```

```
  mapping functions = ([]);
  function `->(string fun)
  {
    if(!functions->fun)
      functions[fun]=lambda(mixed|void ... args)
                     {
                       return sprintf("<FONT COLOR=%s>%{%s %}</FONT>", fun
                     };
    return functions[fun];
  }

// then outside

write(object_of_your_class->chartreuse("stuff"));
```

Nesting Subroutines

```
// Alternate, though identically-behaving, nested subroutine definitions


int outer(int arg)
{
  int x = arg + 35;
  int inner() { return x * 19; };
  return x + inner();
}

// ------------


int outer(int arg)
{
  int x = arg + 35;
  function(void : int) inner = lambda() { return x * 19; };
  return x + inner();
}

// ------------


function outer = lambda(int arg)
  {
    int x = arg + 35;
    return (lambda() { return x * 19; })() + x;
  };

// ------------
```

```
function(int : int) outer = lambda(int arg)
  {
    int x = arg + 35;
    return (lambda() { return x * 19; })() + x;
  };
```

Program: Sorting Your Mail

```
@@INCOMPLETE@@
@@INCOMPLETE@@
```

---

referencesandrecords.html

PLEAC-Pike

---

11. References and Records

Introduction

Taking References to Arrays

Making Hashes of Arrays

Taking References to Hashes

Taking References to Functions

Taking References to Scalars

Creating Arrays of Scalar References

Using Closures Instead of Objects

Creating References to Methods

Constructing Records

Reading and Writing Hash Records to Text Files

Printing Data Structures

Copying Data Structures

Storing Data Structures to Disk

Transparently Persistent Data Structures

Program: Binary Trees

---

packagesetc.html

PLEAC-Pike

---

12. Packages, Libraries, and Modules

Introduction

Defining a Module's Interface

Trapping Errors in require or use

Delaying use Until Run Time

Making Variables Private to a Module

Determining the Caller's Package

Automating Module Clean-Up

Keeping Your Own Module Directory

Preparing a Module for Distribution

Speeding Module Loading with SelfLoader

Speeding Up Module Loading with Autoloader

Overriding Built-In Functions

Reporting Errors and Warnings Like Built-Ins

Referring to Packages Indirectly

Using h2ph to Translate C #include Files

Using h2xs to Make a Module with C Code

Documenting Your Module with Pod

Building and Installing a CPAN Module

Example: Module Template

Program: Finding Versions and Descriptions of Installed Modules

---

---

classesetc.html

---

13. Classes, Objects, and Ties

Introduction

Constructing an Object

Destroying an Object

Managing Instance Data

Managing Class Data

Using Classes as Structs

Cloning Objects

Calling Methods Indirectly

Determining Subclass Membership

Writing an Inheritable Class

Accessing Overridden Methods

Generating Attribute Methods Using AUTOLOAD

Solving the Data Inheritance Problem

Coping with Circular Data Structures

Overloading Operators

Creating Magic Variables with tie

---

dbaccess.html

---

14. Database Access

Introduction

Making and Using a DBM File

Emptying a DBM File

Converting Between DBM Files

Merging DBM Files

Locking DBM Files

Sorting Large DBM Files

Treating a Text File as a Database Array

Storing Complex Data in a DBM File

Persistent Data

Executing an SQL Command Using DBI and DBD

Program: ggh - Grep Netscape Global History

---

15. User Interfaces

Parsing Program Arguments

Testing Whether a Program Is Running Interactively

Clearing the Screen

Determining Terminal or Window Size

Changing Text Color

Reading from the Keyboard

Ringing the Terminal Bell

Using POSIX termios

Checking for Waiting Input

Reading Passwords

Editing Input

Managing the Screen

Controlling Another Program with Expect

Creating Menus with Tk

Creating Dialog Boxes with Tk

Responding to Tk Resize Events

Removing the DOS Shell Window with Windows Perl/Tk

Program: Small termcap program

Program: tkshufflepod

processmanagementetc.html

16. Process Management and Communication

Gathering Output from a Program

Running Another Program

Replacing the Current Program with a Different One

Reading or Writing to Another Program

Filtering Your Own Output

Preprocessing Input

Reading STDERR from a Program

Controlling Input and Output of Another Program

Controlling the Input, Output, and Error of Another Program

Communicating Between Related Processes

Making a Process Look Like a File with Named Pipes

Sharing Variables in Different Processes

Listing Available Signals

Sending a Signal

Installing a Signal Handler

Temporarily Overriding a Signal Handler

Writing a Signal Handler

Catching Ctrl-C

Avoiding Zombie Processes

Blocking Signals

Timing Out an Operation

Program: sigrand

---

sockets.html

PLEAC-Pike

---

17. Sockets

Introduction

```
//------------------------------


//Pike doesn't normally use packed IP addresses. Strings such as "204.148.


// ------------------------------


// DNS lookups can be done with gethostbyname() and gethostbyaddr()

[string host,array ip,array alias] = gethostbyname("www.example.com");
// ip[0] is a string "192.0.32.10"


//------------------------------
```

Writing a TCP Client

```
//------------------------------

Stdio.File sock=Stdio.File();
if (!sock->connect(remote_host,remote_port)) //Connection failed. Error co

{
    werror("Couldn't connect to %s:%d: %s\n",remote_host,remote_port,stren
    return 1;
}
sock->write("Hello, world!"); //Send something to the socket

string answer=sock->read(); //Read until the remote side disconnects. Use

sock->close(); //Not necessary if the sock object goes out of scope here.


//------------------------------
```

Writing a TCP Server

```
//------------------------------

Stdio.Port mainsock=Stdio.Port();
if (!mainsock->bind(server_port))
{
    werror("Couldn't be a tcp server on port %d: %s\n",server_port,strerro
    return 1;
}
while (1)
{
    Stdio.File sock=mainsock->accept();
    if (!sock) break;
    // sock is the new connection

    // if you don't do anything and just let sock expire, the client conne

}
//------------------------------
```

Communicating over TCP

```
//------------------------------

sock->write("What is your name?\n");
string response=sock->read(1024,1); //Reads up to 1KB or whatever is avail

//Buffered reads:

Stdio.FILE sock2=Stdio.FILE(); sock2->assign(sock);
string response=sock2->gets();
//------------------------------


//------------------------------
```

Setting Up a UDP Client

Setting Up a UDP Server

Using UNIX Domain Sockets

Identifying the Other End of a Socket

```
//------------------------------
string other_end=sock->query_address(); //eg "10.1.1.1 123"
//------------------------------
```

Finding Your Own Name and Address

Closing a Socket After Forking

```
//------------------------------
sock->close("r");   //Close the read direction
sock->close("w");   //Close the write direction
sock->close("rw");  //Shut down both directions
sock->close();      //Close completely
//------------------------------
```

Writing Bidirectional Clients

Forking Servers

```
//-----------------------------

//Forking is generally unnecessary in Pike, as the driver works more effic

//-----------------------------
```

Pre-Forking Servers

Non-Forking Servers

```
//-----------------------------

//Incomplete. There's multiple ways to do this, including:

//1) Threaded server (works like forking but clients can share global stat

//2) Multiplexing using select()

//3) Callback mode (puts the sockets under the control of a Backend which

//-----------------------------
```

Writing a Multi-Homed Server

```
//-----------------------------

Stdio.Port mainsock=Stdio.Port();
if (!mainsock->bind(server_port))
{
    werror("Couldn't be a tcp server on port %d: %s\n",server_port,strerro
    return 1;
}
while (1)
{
    Stdio.File sock=mainsock->accept();
```

```
    if (!sock) break;
    string localaddr=sock->query_address(1); //Is the IP address and port

    //The IP will be that of one of your interfaces, and the port should b

}
//-----------------------------
```

Making a Daemon Server

```
//-----------------------------

if (!System.chroot("/var/daemon")) werror("Unable to chroot to /var/daemon
//Incomplete (I don't fork in Pike). See predef::fork() and Process.create

//-----------------------------
```

Restarting a Server on Demand

```
//-----------------------------

//The best way to restart the server is to adopt a microkernel concept and

//the server that need updating. However, if you must reload, see Process.

//-----------------------------
```

Program: backsniff

Program: fwdport

18. Internet Services

Simple DNS Lookups

Being an FTP Client

Sending Mail

Reading and Posting Usenet News Messages

Reading Mail with POP3

Simulating Telnet from a Program

Pinging a Machine

Using Whois to Retrieve Information from the InterNIC

Program: expn and vrfy

---

cgiprogramming.html

---

19. CGI Programming

Introduction

Writing a CGI Script

Redirecting Error Messages

Fixing a 500 Server Error

Writing a Safe CGI Program

Making CGI Scripts Efficient

Executing Commands Without Shell Escapes

Formatting Lists and Tables with HTML Shortcuts

Redirecting to a Different Location

Debugging the Raw HTTP Exchange

Managing Cookies

Creating Sticky Widgets

Writing a Multiscreen CGI Script

Saving a Form to a File or Mail Pipe

Program: chemiserie

20. Web Automation

Introduction

Fetching a URL from a Perl Script

Automating Form Submission

Extracting URLs

Converting ASCII to HTML

Converting HTML to ASCII

Extracting or Removing HTML Tags

Finding Stale Links

Finding Fresh Links

Creating HTML Templates

Mirroring Web Pages

Creating a Robot

Parsing a Web Server Log File

Processing Server Logs

Program: htmlsub

Program: hrefsub

Prev	Home	Next
CGI Programming		Helpers

---

a1102.html

PLEAC-Pike

Prev

A. Helpers

```
// (this section is optional; use it if you need to import very
// generic stuff for the whole code)
//
// Note: To avoid clutter each example will only include any necessary
// code. However, it should be understood that:
//
// * The following constants need to be defined:
//
//    constant FALSE = 0, TRUE = 1, PROBLEM = 1, OK = 0,
//      EOF = -1, NULL = "", NEWLINE = "\n", LF = 10, SPACE = 32;
//
// * Each example needs to be enclosed within the following block:
//
//    int main(int argc, array(string) argv)
//    {
//      ...
//    }
//
//   where a 'main' is not provided. Also:
//
//   - Any function definitions would ordinarily be placed
//     before, and outside of, 'main'
//   - Variables can be assumed to be locals residing in 'main';
//     any 'global' variables will be defined at the start of the
```

```
//     code example prior to any function definitions


// ----------------------------


string chop(string s, void|int size)
{
  int length = sizeof(s);
  return size > 0 && size < length ? s[..length - (size + 1)] : s;
}
```
