Introduction comparative à bash , perl , python et ruby

$Id: intro-bppr.fodp 574 2012-05-21 09:50:04Z aicardi $

S. Aicardi, J. Charbonnel, D. Delavennat, B. Métrot

ANF Mathrice – Angers, mai 2012

Variables et expressions

Variables scalaires

```
typeset -i num     # Déclaration d'une variable
                    Numérique

num=1              # Affectations
s="bye"

typeset -i num=1  # Déclaration + affectation
                      d'une variable numérique

# Chaine de caractères
s2="Bye $s world"      # Bye bye world
s2="Bye $num world"    # Bye 1 world
s2='Bye $s world'      # Bye $s world
```

variables scalaires

```
my $i, $s ;                     # déclarations
$i = 1 ;                    # affectation
$s = "bye" ;

my $i = 1 ;                         # déclaration + initialisation
my $s = "bye" ;

my ($i,$s) = (1,"bye") ;        # déclaration + initialisation

# chaînes de caractères

$s2 = "Bye ".$s." world" ;  # Bye bye world
$s2 = "Bye $s world" ;      # Bye bye world
$s2 = 'Bye $s world' ;      # Bye $s world
```

variables scalaires

```
i = 1
s1 = "bye"
i,s1 = 1,"bye"

s2 = "Bye "+s1+" world"                 # Bye bye world
s2 = "Bye %s world"%s1                  # Bye bye world
s2 = "Bye {0} world".format(s1)         # Bye bye world
```

variables scalaires

```
i = 1
s = "bye"

i,s = 1,"bye"

s2 = "Bye "+s+" world"  # Bye bye world
s2 = "Bye #{s} world"   # Bye bye world
s2 = 'Bye s world'      # Bye s world
```

pragma

```
use warnings ;          # équivalent à : #!/usr/bin/perl -w

use strict ;            # oblige les déclarations dans la suite

no warnings ;           # annule use warnings
no strict ;             # annule use strict
```

modules

```
use Data::Dumper ;
use File::Handle ;
```

```
import ldap.sasl
from mymodule import *
```

```
require 'rubygems'
require 'pp'
```

tableau

```
typeset -a l1=(1 2.1 "trois")

echo ${#l1[@]}     # Nb d'éléments => 3
echo ${l1[@]}      # 1 2.1 trois

l1[1]=4            # 1 4 trois
```

tableaux et listes

```
my @l1 = (1, 2.1, "trois") ;
print scalar(@l1) ;                 # 3
for $i (@l1) { print "$i " }        # 1 2.1 trois

$l1[1] = 4 ;                        # 1 4 trois

pop @l1 ; # 1 4
push @l1,('trois',5) ;              # 1 4 trois 5

shift @l1 ;                         # 4 trois 5
unshift @l1,"six" ;                 # six 4 trois 5

($a,$b,@l3) = @l1 ;                 # $a=six, $b=4, @l3=(trois,5)

@l2 = ( "a",@l1,"b" ) ;             # a six 4 trois 5 b
```

```
use Data::Dumper ;
print Dumper(@l1,@l2) ;
```

tableaux et listes

```
@l1 = (4,"trois",5) ;
@l2 = ( "a",@l1,"b" ) ;            # a 4 trois 5 b
@l2 = ( "a",\@l1,"b" ) ;           # a \(4 trois 5) b
```

```
use Data::Dumper ;
print Dumper(@l1,@l2) ;
```

listes

```
l1 = [ 1, 2.1, "trois" ]   #liste
print(len(l1))             # → 3

for i in l1:
  print i,                      # → 1 2.1 trois

l1[1] = 4                     # → 1 4 trois
print(l1[:1])                 # → [1]
print(l1[1:])                 # → [4,'trois']

l1.pop()                      # [1, 4]
l1.append(['trois',5])        # [1,4,['trois',5]]
l1.pop()                      # [1, 4]
l1.extend(['trois',5])        # [1,4,'trois',5]


del l1[0]                     # → [4,'trois',5]
l1.insert(0,"six")            # → ['six',4,'trois',5]

a,b,l3 = l1
ValueError: too many values to unpack
a,b,l3 = l1[0],l1[1],l1[2:]   # a → 'six', b → 4, l3 → ['trois',5]

l2 = [ "a",l1,"b" ]           # → [ 'a', ['six', 4, 'trois', 5], 'b']
```

tableaux (ou *tuple*)

```
T1 = ( 1, 4, "trois" )      #tableau
print(T1[:1])               # (1,)
print(T1[1:])               # (4,'trois')

# un tableau n'est pas une liste !
T1[1]=2.1
'tuple' object does not support item assignment

T1.pop()
'tuple' object has no attribute 'pop'
```

tableaux et listes

```
l1 = [ 1, 2.1, "trois" ]

l1.each{|i| print "#{s} " }      # 1 2.1 trois

l1[1] = 4                        # l1 → [ 1, 4, "trois" ]

l1.pop                           # l1 → [ 1, 4 ]

l1.push('trois',5)               # l1 → [ 1, 4, "trois", 5 ]

l1.shift                         # l1 → [ 4, "trois", 5 ]

l1.unshift("six")                # l1 → [ "six", 4, "trois", 5 ]

a,b,l3 = l1                      # a="six", b=4, l3="trois"

l2 = ["a",l1,"b" ]               # l2 → [ "a", ["six",4,"trois",5], "b" ]
```

```
require 'pp'

l1 = [ 1, 2.1, "trois" ]
pp l1          #STDOUT → [ 1, 2.1, "trois" ]
```

hashes (tableau associatif)

```
typeset -A h[3]          # Déclaration

h[z]=1                   # Affectation
h[y]="deux"
h[x]=3.0

for k in x y z           # Parcours + affichage
do
  echo ${h[$k]}
done

unset h[y]               # Suppresion d'un élément
```

hashes

```
my %h = {z=>1, y=>"deux", x=>3.0 } ;

$h{1} = 4 ;                  # y=>"deux" z=>1 x=>3.0 1=>4
$h{y} = 2 ;                  # y=>2 z=>1 x=>3.0 1=>4

delete $h{z} ;               # y=>2 x=>3.0 1=>4

@k = keys %h ;              # (y,x,1)

@v = values %h ;            # (2,3.0,4)

for $k (keys %h) { print "($k→$h{$k}) "; }       # (y→2) (x→3.0) (1→4)
```

```
use Data::Dumper ;
print Dumper %h ;
```

hashes

```
h = {'z':1, 'y':"deux", 'x':3.0 }

h[1] = 4          # 'y':"deux", 'z':1 'x':3.0 1:4
h['y'] = 2       # 'y':2 'z':1 'x':3.0 1:4

del h['z']       # 'y':2 'x':3.0 1:4

k = h.keys()     # vue des clés : à ce moment de l'exécution → ['y', 'x', 1]

v = h.values()   # vue des valeurs : à ce moment de l'exécution → [2, 3.0, 4]

for k in h.keys():
  print "(%s->%s)"%(k,h[k]),      # (y->2) (x->3.0) (1->4)
```

```
print(h)
```

hashes

```
h = { :z=>1, :y=>"deux", :x=>3.0 }

h[1]  = 4          # {:y=>"deux",:z=>1,:x=>3.0,1=>4}
h[:y] = 2          # {:y=>2,:z=>1,:x=>3.0,1=>4}

h.delete(:z)       # {:y=>2,:x=>3.0,1=>4}

k = h.keys         # k → [:y,:x,1]
v = h.values       # v → [2,3.0,4]

h.each{|k,v| print "(#{k}->#{v}) "}      # (y->2) (x->3.0) (1->4)
```

```
require 'pp'

h = { :z=>1, :y=>"deux", :x=>3.0 }
pp h       # STDOUT → { :z=>1, :y=>"deux", :x=>3.0 }
```

références

```
@l1 = (4,"trois",5) ;
@l2 = ( "a",@l1,"b" ) ;            # a 4 trois 5 b
@l2 = ( "a",\@l1,"b" ) ;           # a \(4 trois 5) b
```

```
my $l1 = [1, 2.1, "trois"] ;
for $i (@$l1) { print "$i " }

$l1->[1] = 4 ;                     # 1 4 trois
@l2 = ( "a",$l1,"b" ) ;            # a \(1 4 trois) b

my $h = [z=>1, y=>"deux", x=>3.0] ;
```

```
$h->{y} = 2 ;                 # y=>2 z=>1 x=>3.0

@k = keys %$h ;              # (y,x,x)

@v = values %$h ;            # (2,1,3.0)

for $k (keys %$h)
   { print "($k→$h->{$k}) "; }    # (y→2) (x→3.0) (z→1)
```

expression régulières

```
s=" a = 5 "

if echo "$s" | grep -q -e "^[[:space:]]*[[:alnum:]]+[[:space:]]*=
[[:space:]]*[[:alnum:]]+[[:space:]]*$"
then
  a=`echo "$s" | sed 's|^[[:space:]]*\([a-Z]\+\)[[:space:]]*=
    [[:space:]]*[0-9]\+[[:space:]]*$|\1|'`
  b=`echo "$s" | sed 's|^[[:space:]]*[a-Z]\+[[:space:]]*=
    [[:space:]]*\([0-9]\+\)[[:space:]]*$|\1|'`
fi

echo "$a , $b"            # Affiche a , 5
```

expressions régulières

```
$s = " a = 5 " ;

if ($s =~ /^\s*\S+\s*=\s*\S+\s*$/) { … }

if ($s =~ m|^\s*\S+\s*=\s*\S+\s*$|) { … }

($a,$b) = $s=~/^\s*(\S+)\s*=\s*(\S+)\s*$/ ;      # $a → "a", $b → "5"

$s =~ s/\s//g ;                                  # $s → "a=5"
```

expressions régulières

```
import re
s = " a = 5 "

if (re.search('^\s*\S+\s*=\s*\S+\s*$/',s):

m=re.search('\s*(\S+)\s*=\s*(\S+)\s*$',s)
x,y = m.group(1,2)                              # x → "a", y → "5"

s = re.sub('\s','',s)                           # s → "a=5"
```

expressions régulières

```
s = " a = 5 "

if m=s.match(/^\s*\S+\s*=\s*\S+\s*$/) then
 p m
end

m = s.match(/^\s*(\S+)\s*=\s*(\S+)\s*$/)
                        # m → #<MatchData " a = 5 " 1:"a" 2:"5">
                        # m[1],m[2] → a,5

s.gsub!(/\s/,'')        # s → "a=5"
```

Structures de contrôle

instructions conditionnelles

```
if command-test
then
    command1
else
    command2
fi
```

```
if command-test1
then
    command1
elif command-test2
then
    command2
elif command-test3
then
    command3
else
    command4
fi
```

```
case $var in
value1 )
    command1
    ;;
value2 )
    Command2
    ;;
* )
    command2
esac
```

```
# Exemples de command-test

[ "$str" = "chaine" ]     # Vrai si $str vaut chaine

[ $N -eq 10 ]          # Vrai si $N vaut 10

[ $N -gt 10 ]          # Vrai si $N supérieur à 10 (strictement)

[ $N -lt 10 ]          # Vrai si $N inféieur à 10 (strictement)
```

instructions conditionnelles

```
if (…) {…}

if (defined($i))
{
  print $i ;
}


… if …

print $i if defined($i)
```

```
unless (…) {…}

unless ($i==0)
{
  print $i ;
}


… unless …

print $i unless $i==0
```

```
if (…) {…}
else {…}


if (…) {…}
elsif (…)
else {…}
```

instructions conditionnelles

```
if ...:
  ...
elif:
  ...
else:
  ...
```

instructions conditionnelles

```
if … [then] … [else] end

… if …

print i if ! i.nil?
```

```
unless … [then] … [else] end

… unless …

print i unless i.nil?
```

```
if … then … else … end

if … then … elsif … else … end
```

instructions itératives

```
for var in arg1 … argN
do
    command1
    …
    commandN
done
```

```
for (( expr1 ; expr2 ; expr3 ))
do
    command1
    …
    commandN
done
```

```
while command-test
do
    command1
    …
    commandN
done
```

```
until command-test
do
    command1
    …
    commandN
done
```

instructions itératives

```
while (…) {…}

… while …

print $i while $i<100 ;


do {…} while …
```

```
for (… ; … ; …) {…}


for $var (@liste) {…}
```

```
while (…)
{
  …
  next if …
  …
  last if …
}
```

instructions itératives

```
while ...:
  ...

for var in liste:
  ...

for var in range(10):
  ...
```

```
while ...:
  ...
  if ...:
    continue   # itération suivante
  if ...:
    break      # fin des itérations
```

instructions itératives

```
while (...) {...}

... while ...

print i while i<100

do {...} while ...
```

```
for (... ; ... ; ...) {...}

for $var (@liste) {...}
```

```
while (...)
{
  ...
  next if ...
  ...
  last if ...
}
```

Fonctions

paramètres

```
function f () {
    # Nombre de paramètres
    echo $#

    # Tous les paramètres
    echo $*

    # Les paramètres 1 à un
    echo $1
    echo $2
    echo $3
}

# Appel de fonction
f "$s" 10 "arg"
```

paramètres

```
sub f
{
  ($a,$b) = @_ ;
  …
}

f(1.2,"tagada") ;
```

```
sub f
{
  my (@p1,@p2) = @_ ;

  # @p1 → (1,2,3,4,5)
  # @p2 → ()
}

@l1 = (1,2,3) ; @l2 = (4,5) ;
f(@l1,@l2) ;
```

```
sub f
{
  my ($p1,$p2) = @_ ;

  # $p1 → [1,2,3]
  # $p2 → [4,5]
}

@l1 = (1,2,3) ; @l2 = (4,5) ;
f(\@l1,\@l2) ;
```

paramètres

```
def f(a,b):
  ...

f(1.2,"tagada")
```

```
def f(p1,p2=[]):
  # p1 → [[1, 2, 3], [4, 5]]
  # p2 → []


l1 = [1,2,3]
l2 = [4,5]
f([l1,l2])
```

```
def f(p1,p2=[]):
  # p1 → [1, 2, 3]
  # p2 → [4,5]


l1 = [1,2,3]
l2 = [4,5]
f(l1,l2)
```

paramètres

```
def f(a,b)
  # a → 1.2
  # b → "tagada"
end

f(1.2,"tagada")
```

```
def f(p1,p2=nil)
  # p1          → [[1,2,3],[4,5]]
  # p1.flatten → [1,2,3,4,5]
  # p2          → nil
end

l1,l2 = [1,2,3],[4,5]
f([l1,l2])
```

```
def f(p1,p2=nil)
  # p1 → [1,2,3]
  # p2 → [4,5]
end

l1,l2 = [1,2,3],[4,5]
f(l1,l2)
```

retour

```
function f () {
  ...
  return -1
  ...
}


# Code de retour stocké dans variable $?
f
RETCODE=$?  # $RETCODE vaut -1
```

```
function f () {
  command1
  ...
  commandN
}

# Code de retour stocké dans variable $?
f
RETCODE=$?  # $RETCODE contient le code de
            # retour de commandN
```

retour

```
sub f
{
  …
  return -1 ;
  …
}

$i = f() ; # $i → -1
```

```
sub f
{
  …
  -1 ;        # dernière instruction évaluée
}

$i = f() ; # $i → -1
```

retour

```
def f():

  ...
  return -1
  ...


i = f() ; # i → -1
```

retour

```
def f
  ...
  return -1
  ...
end

i = f() # i → -1
```

```
def f
  ...
  -1    # dernière instruction évaluée
end

i = f() # i → -1
```

Fichiers

```
# Lecture sur l'entrée standard
while read LINE
do
  echo $LINE
done
```

```
# Lecture du fichier /etc/passwd ligne par ligne
while read LINE
do
 # N'affiche que les lignes non commenté
 echo "$LINE" | grep -v "^#"
done < /etc/passwd
```

```
# Ecriture en écrasant le fichier
echo "Hello $myvar World !" > /tmp/file.txt

# Ecriture en ajoutant en fin de fichier
Echo "Hello $myvar World !" >> /tmp/file.txt
```

```perl
while (<>)  # lit sur stdin
{
  print ;   # équivalent à: print $_ ;
}
```

```perl
use FileHandle ;

my $fh = new FileHandle("/etc/passwd") or die $! ;

while (<$fh>)
{
  chomp ;               # supprime le \n final
  s/#.*$// ;            # supprime les commentaires
  next unless /\S/ ;  # ignore ligne vide
  …
}
$fh->close() ;
```

```perl
use FileHandle ;

my $fh = new FileHandle(">/tmp/toto") or die $! ;

for $l (@l)
{
  $fh->printf("%d: %s\n",$i++,$l) ;
}
$fh->close() ;
```

```
import sys

for line in sys.stdin:    # lit sur stdin
  print line,
```

```
import re

fh = open("/etc/passwd","r")

for line in fh:
  line=re.sub('#.*$','',line.rstrip('\n'))
      # supprime les commentaires
  if re.match("\s*$",line):
    continue
    # ignore ligne vide
  ...

fh.close()
```

```
fh = open("/tmp/toto", "w")

i=0
for line tableau:
  i+=1
  fh.write("%d: %s\n",i,line)

fh.close()
```

```
while gets    # lit sur stdin
{
  print       # équivalent à: print $_ ;
}
```

```
File.open("/etc/passwd",'r'){|fin|
 File.open("/tmp/passwd",
      File::WRONLY|File::TRUNC|File::CREAT,
      0600
 ){|fout|
  fin.each_line{|line|
    line.chomp!             # supprime le \n final
  s/#.*$// ;                # supprime les commentaires
  next unless /\S/ ;        # ignore ligne vide
   …
   fout.print line
  }
}
```

```
File.open("/tmp/toto",'r'){|file|
  file.each{|line|
    puts "#{file.lineno} : #{line}"
  }
}
```

POO

classes

```
# fichier Url.pm

package Url ;
require Exporter ;
our @ISA = qw(Exporter);
our @EXPORT=qw() ;

sub new                          # constructeur
{
  my($type,$s…) = @_ ;
  my($this) ;          # le futur objet

  $this->{url} = $s ;      # initialisation
  $this->{…} = … ;

  bless $this,$type ;
  return $this ;
}

sub get
{
  my ($this) = @_ ;
  return wget($this->{url}) ;
}
```

classes

```
# fichier Url.py

import urllib
class Url:

  def __init__(self,uri):  #constructeur
    self.uri=uri           #initialisation

  def get(self):           #méthode
    print urllib.urlopen(self.uri).read()
```

classes

```
# fichier custom-uri.rb
require 'openuri'

class CustomURI
  attr_reader :uri     #accesseur

  def initialize(uri)  #constructeur
    @uri = uri         #initialisation
  end
  def get
     open(@uri)
  end
end
```

objets

```
use Url ;

my $u = new Url("http://www.insmi.fr",…);
$u->get() ;
```

objets

```
from Url import *

u = Url("http://www.insmi.fr")
u.get()
```

objets

```
require "custom-uri"

u = CustomURI.new("http://www.insmi.fr")
u.get
puts u.uri       # STDOUT → http://www.insmi.fr
```