```
> #================================================
> #  review of loops  (including some lab 6 problems)
> #================================================
> # Compute the set of numbers in 1..n that divide n;
  factorlist := proc(n :: integer)
  local divisor;
  divisor := (n, i) → piecewise(frac(n/i) = 0, i, frac(n/i) ≠ 0, -100);
  return {seq(divisor(n, i), i = 1 ..n)} minus {-100};
  end proc;
```

```
factorlist := proc(n::integer)                                          (1)
  local divisor;
  divisor := (n, i) → piecewise(frac(n/i) = 0, i, frac(n/i) <> 0, -100);
  return minus({seq(divisor(n, i), i = 1 ..n)}, {-100})
end proc
```

```
>
> # Write a procedure that takes a list of numbers and returns true if the list contains an arithmetic series and false
      otherwise. Recall that an arithmtic series is one where the difference between any two successive terms is
      equal. E.g. 2,6,10,14,18 is an arithmetic series.
  #  Your procedure must work for lists of length 1 and 2 as well.
> q2 := proc(L :: list)
  local num, cdiff;
  if nops(L) ≤ 2 then return true
  else
  cdiff := L[2] - L[1];
  for num from 3 to nops(L) do
  if L[num] -L[num - 1] ≠ cdiff then return false;
  end if;
  end do;
  return true;
  end if;
  end proc;
```

```
q2 := proc(L::list)                                                     (2)
  local num, cdiff;
  if nops(L) <=2 then
      return true
  else
      cdiff := L[2] - L[1];
      for num from 3 to nops(L) do
          if L[num] - L[num - 1] <> cdiff then return false end if
      end do;
      return true
  end if
end proc
```

```
>
  #Write a procedure that takes a list and returns the absolute difference of the sum of the odd elements and the sum
      of the even elements.
   #For example, when called on the list [1, 2, 3, 4], it should# return abs(4-6) = 2.


> sumDiff := proc(L :: list)
```

```
    local num, oddsum, evensum;
    oddsum := 0;
    evensum := 0;
    for num from 1 to nops(L) do
    if irem(num, 2) = 0 then evensum := evensum + L[num];
    else oddsum := oddsum + L[num];
    end if;
    end do;
    return abs(oddsum - evensum);
    end proc;
sumDiff := proc(L::list)                                                    (3)
    local num, oddsum, evensum;
    oddsum := 0;
    evensum := 0;
    for num to nops(L) do
        if irem(num, 2) = 0 then evensum := evensum + L[num] else oddsum := oddsum + L[num] end if
    end do;
    return abs(oddsum - evensum)
end proc
```

```
> #Write a procedure that takes a list and a positive integer n and returns a list containing n concatenated copies of
      the given list.
```

```
> listcopy := proc(L :: list, n :: posint)
    local num, L2;
    L2 := [ ];
    for num from 1 to n do
    L2 := [op(L2), op(L)];
    end do;
    return L2
    end proc;
listcopy := proc(L::list, n::posint)                                        (4)
    local num, L2;
    L2 := [ ]; for num to n do L2 := [op(L2), op(L)] end do; return L2
end proc
```

```
> #=====================================================
> #          recursive programs
> #=====================================================
>
> # Printing n,n-1,...,1 without using a loop  (or seq or map ... )
> recprint := proc(n :: posint)
    print(n);
    if n > 1 then recprint(n - 1); end if;
    end proc;
    recprint := proc(n::posint) print(n); if 1 < n then recprint(n - 1) end if end proc      (5)
> recprint(5);
                                    5
                                    4
                                    3
                                    2
                                    1                                       (6)
```

```
>
>
>
>
> # Computing an integer power using multiplications and no loops
>
> recpower := proc(x :: numeric, n :: posint)
    if n = 1 then return x;
    else return x·recpower(x, n − 1);
    end if;
   end proc;
```

recpower := **proc**(*x*::*numeric*, *n*::*posint*) **(7)**

if *n* = 1 **then return** *x* **else return** *x* * *recpower*(*x*, *n* − 1) **end if**

end proc

```
>
> recpower(2, 4);
```

 (8)

```
>
>
>
```