```
> #==========================================
  #  Mar 21. Examples of recursive and iterative programs
> #==========================================
>
> # number of digits in an integer (recursive)
> recnumdig :=proc(n :: posint)
   if n < 10 then return 1;
   else return 1 + recnumdig(trunc(n/10));
   end if;
   end proc;
```

```
recnumdig := proc(n::posint)                                                    (1)
    if n < 10 then return 1 else return 1 + recnumdig(trunc(1/10*n)) end if
end proc
```

```
> recnumdig(10000);
                                        5                                       (2)
> #==========================================
  # printing a (palindromic sequence
> #==========================================
>
> rec1 :=proc(n :: integer)
   if n = 1 then printf(" %d ", n);
   else printf(" %d ", n);
        rec1(n - 1);
        printf(" %d ", n);
   end if;
   end proc;
```

```
rec1 := proc(n::integer)                                                        (3)
    if n = 1 then printf(" %d ", n) else printf(" %d ", n); rec1(n - 1); printf(" %d ", n)
    end if
end proc
```

```
> rec1(10);
 10  9  8  7  6  5  4  3  2  1  2  3  4  5  6  7  8  9  10
>
> #=============recursive max ============================
>
    mymax := (a, b)→piecewise( a > b , a, a ≤ b, b);
                          mymax := (a, b)→piecewise(b < a, a, a ≤ b, b)         (4)
> L := [1, 34, 45, 12, 34, 6, 234, 6, 78, 234, 2];
                             L := [1, 34, 45, 12, 34, 6, 234, 6, 78, 234, 2]    (5)
> recmax :=proc(k :: integer) #return the max of L[1..k]
   if k = 1 then return L[1];
   else return mymax(L[k], recmax(k - 1));
   end if;
   end proc;
```

```
recmax := proc(k::integer)                                                      (6)
```

```
    if k = 1 then return L[1] else return mymax(L[k], recmax(k - 1)) end if
end proc
```

```
> recmax(nops(L));
```

$$234 \tag{7}$$

```
> #========sqrt(1 + sqrt(1 + .... sqrt(1 +x)....))====================
```

```
> rec2 := proc(x :: numeric, n :: posint)
    if n = 1 then return x;
    else return sqrt(1 + evalf(rec2(x, n - 1)));
    end if;
    end proc;
```

```
rec2 := proc(x::numeric, n::posint)
    if n = 1 then return x else return sqrt(1 + evalf(rec2(x, n - 1))) end if
end proc
```

(8)

```
> rec2(3, 4);
```

$$1.652891650 \tag{9}$$

```
> L := [seq(rec2(3, i), i = 1..30)];
```

$$L := [3, 2.000000000, 1.732050808, 1.652891650, 1.628769981, 1.621348199, 1.619057812, 1.618350337, 1.618131743, 1.618064196, 1.618043323, 1.618036873, 1.618034880, 1.618034264, 1.618034074, 1.618034015, 1.618033997, 1.618033991, 1.618033989, 1.618033989, 1.618033989, 1.618033989, 1.618033989, 1.618033989, 1.618033989, 1.618033989, 1.618033989, 1.618033989, 1.618033989, 1.618033989] \tag{10}$$

```
>
> #================some symbolic math=============
> # what is f(f(....f(x)...)) when f(x) =  (1-x)^-1?
>
> recfunc := proc(n :: posint)
    if n = 1 then return 1/(1 - x);
    else return simplify(1/(1 - recfunc(n - 1)));
    end if;
    end proc;
```

```
recfunc := proc(n::posint)
    if n = 1 then return 1/(1 - x) else return simplify(1/(1 - recfunc(n - 1))) end if
end proc
```

(11)

```
> recfunc(10);
```

$$-\frac{1}{-1+x} \tag{12}$$

```
> L := [seq(recfunc(i), i = 1..10)];
```

$$L := \left[\frac{1}{1-x}, \frac{-1+x}{x}, x, -\frac{1}{-1+x}, \frac{-1+x}{x}, x, -\frac{1}{-1+x}, \frac{-1+x}{x}, x, -\frac{1}{-1+x}\right] \tag{13}$$

```
>
> recfunc(100000);
Error, (in recfunc) too many levels of recursion
> # Use the pattern to solve the problem!
> recfunc(irem(1000000, 3));
```

$$\frac{1}{1-x} \tag{14}$$

```
>
```