```
> #FOR LOOP EXAMPLES
> #1. Search for an element in a list
>
> myListSearch :=proc(L :: list, x :: numeric)
   local num;
   for num from 1 to nops(L) do
   if L[num]=x then return num;
   end if;
   end do;
   print("Element not found");
   return -1;
   end proc;
```

```
myListSearch := proc(L::list, x::numeric)                    (1)
    local num;
    for num to nops(L) do if L[num]=x then return num end if end do;
    print("Element not found");
    return -1
end proc
```

```
> L := [1, 1, 2, 3, 4];
```

$$L := [1, 1, 2, 3, 4] \tag{2}$$

```
> myListSearch(L, 1);
```

$$1 \tag{3}$$

```
> a := myListSearch(L, 5); a;
```

$$\text{"Element not found"}$$

$$a := -1$$

$$-1 \tag{4}$$

```
> #Make a more general version of searching a list by adding afunction for matching
  matchfunc :=proc(x :: numeric, c :: numeric)
   if abs(x - c) < 0.05 then return true;#instead of exact matches allow some mismatch
   else return false;
   end if;
   end proc;
```

```
matchfunc := proc(x::numeric, c::numeric)                    (5)
    if abs(x - c) < 0.05 then return true else return false end if
end proc
```

```
> myListSearch2 :=proc(L :: list, x :: numeric)
   local num;
   for num from 1 to nops(L) do
   if matchfunc(L[num], x) = true then return num;
   end if;
   end do;
   print("Element not found");
   return -1;
   end proc;
```

```
myListSearch2 := proc(L::list, x::numeric)                   (6)
    local num;
```

```
    for num to nops(L) do if matchfunc(L[num], x) = true then return num end if end do;
    print("Element not found");
    return -1
end proc
```

```
> myListSearch2(L, 3.99);
```

5 **(7)**

```
> L := [1, 2, 3, 1, 1];
```

L := [1, 2, 3, 1, 1] **(8)**

```
> #count number of occurrences of a number in a list
> numOccur :=proc(L :: list, x :: numeric)
    local num, count;
    count := 0;
    for num from 1 to nops(L) do
    if L[num] = x then count := count + 1;
    end if;
    # print("CURRENT COUNT VALUE:", count);
    end do;
    return count;
    end proc;
numOccur := proc(L::list, x::numeric)                                    (9)
    local num, count;
    count := 0;
    for num to nops(L) do if L[num] = x then count := count + 1 end if end do;
    return count
end proc
```

```
> numOccur(L, 5);
```

0 **(10)**

```
> numOccur(L, 1);
```

3 **(11)**

```
> #adding the elements of a list
>
> sumList :=proc(L :: list)
    local num, sum;
    sum := 0;
    for num from 1 to nops(L) do
    sum := sum + L[num];
    end do;
    return sum;
    end proc;
sumList := proc(L::list)                                                 (12)
    local num, sum;
    sum := 0; for num to nops(L) do sum := sum + L[num] end do; return sum
end proc
```

```
> sumList(L);
```

11 **(13)**

```
> #Exercise: modify the above to get the mean, min, max of a list
>
> #delete all elements <= x from a list
> modifyList := proc(L :: list, x :: numeric)
    local L2, num;
    L2 := [ ];
    for num from 1 to nops(L) do
    if L[num] > x then L2 := [op(L2), L[num]]; end if;
    end do;
    return L2;
    end proc;
```

```
modifyList := proc(L::list, x::numeric)                                    (14)
    local L2, num;
    L2 := [ ];
    for num to nops(L) do if x < L[num] then L2 := [op(L2), L[num]] end if end do;
    return L2
end proc
```

```
> L := [1, 2, 2, 3, 4, 4];
  modifyList(L, 2);
```

$$L := [1, 2, 2, 3, 4, 4] \tag{15}$$

$$[3, 4, 4]$$

```
> modifyList(L, 10);
```

$$[\,] \tag{16}$$

```
> #reversing a list
> revList := proc(L :: list)
  local L2, num;
    L2 := L;
    for num from 1 to nops(L) do
  L2[nops(L) - num + 1] := L[num];
    end do;
    return L2;
    end proc;
```

```
revList := proc(L::list)                                                   (17)
    local L2, num;
    L2 := L; for num to nops(L) do L2[nops(L) - num + 1] := L[num] end do; return L2
end proc
```

```
> revList(L);
```

$$revList([1, 2, 3, 1, 1]) \tag{18}$$

```
> L;
```

$$[1, 2, 2, 3, 4, 4] \tag{19}$$

```
> revList([ ]);
```

$$[\,] \tag{20}$$

```
> #reversing in place
    L := [1, 2, 3, 4, 5, 6];
```

$$L := [1, 2, 3, 4, 5, 6] \tag{21}$$

```
> for num from 1 to trunc( nops(L)/2 ) do
  L[nops(L) - num + 1] := L[num];
  end do;
```

$$L_6 := 1$$
$$L_5 := 2$$
$$L_4 := 3 \quad \textbf{(22)}$$

```
> L;
```

$$[1, 2, 3, 3, 2, 1] \quad \textbf{(23)}$$

```
> #try again
> L := [1, 2, 3, 4, 5, 6];
```

$$L := [1, 2, 3, 4, 5, 6] \quad \textbf{(24)}$$

```
> for num from 1 to trunc( nops(L)/2 ) do
  tmp := L[nops(L) - num + 1];
  L[nops(L) - num + 1] := L[num];
  L[num] := tmp;
  end do;
```

$$tmp := 6$$
$$L_6 := 1$$
$$L_1 := 6$$
$$tmp := 5$$
$$L_5 := 2$$
$$L_2 := 5$$
$$tmp := 4$$
$$L_4 := 3$$
$$L_3 := 4 \quad \textbf{(25)}$$

```
> L
```

$$[6, 5, 4, 3, 2, 1] \quad \textbf{(26)}$$

```
> L := [1, 2]; ll := [1, 2, 3];
```

$$L := [1, 2]$$
$$ll := [1, 2, 3] \quad \textbf{(27)}$$

```
> evalb(L = ll);
```

$$false \quad \textbf{(28)}$$

```
> #Is a list palindromic (i.e. L = revList(L))?
> Pdrome :=proc(L :: list)
  local num, len, result;
  len := nops(L);
  result := true;
  for num from 1 to trunc( len/2 ) do
  if L[len - num + 1] ≠ L[num] then result := false;
  end if;
```

```
    #print(num, result);
     end do;
     return result;
     end proc;
Pdrome := proc(L::list)                                                    (29)
    local num, len, result;
    len := nops(L);
    result := true;
    for num to trunc(1/2*len) do
        if L[len - num + 1] <> L[num] then result := false end if
    end do;
    return result
end proc
> L;
                                    L                                      (30)
> L := [1, 2, 3, 2, 1]; L2 := [1, 2, 3, 4, 5, 6];
                                 L := [1, 2, 3, 2, 1]
                              L2 := [1, 2, 3, 4, 5, 6]                     (31)
> Pdrome(L);
                                  true                                     (32)
> Pdrome(L2);
                                  false                                    (33)
>
>
>
>
>
>
```