```
> #Lab 8 Thu solutions
>
> #Q1
>
> goldbach := proc(a :: posint)
   local num, L;
   if irem(a, 2) = 1 then print("Error: input is an odd number"); return [ ];
  else
   for num from 2 to a − 2 do
   if isprime(num) = true and isprime(a − num) = true then
   L := [num, a − num];
   return L;
   end if;
   end do;
   print("The Goldbach conjecture is resolved!");
   return [ ];
   end if;
   end proc;
```

```
goldbach := proc(a::posint)                                          (1)
    local num, L;
    if irem(a, 2) = 1 then
        print("Error: input is an odd number"); return [ ]
    else
        for num from 2 to a − 2 do
            if isprime(num) = true and isprime(a − num) = true then
                L := [num, a − num]; return L
            end if
        end do;
        print("The Goldbach conjecture is resolved!");
        return [ ]
    end if
end proc
```

```
> goldbach(18);
                                [5, 13]                                  (2)
>
> #Q2
> pyth := proc( )
   local num;
   for num from 1 to 16 do
   if frac(sqrt(16^2 + num^2)) = 0 then print(num, 16, sqrt(16^2 + num^2));
   end if;
   end do;
   for num from 17 to 1000 do
   if frac(sqrt(16^2 + num^2)) = 0 then print(16, num, sqrt(16^2 + num^2));
   end if;
   end do;
```

```
    end proc;
pyth := proc( )                                                    (3)
    local num;
    for num to 16 do
        if frac(sqrt(256 + num^2)) = 0 then print(num, 16, sqrt(256 + num^2)) end if
    end do;
    for num from 17 to 1000 do
        if frac(sqrt(256 + num^2)) = 0 then print(16, num, sqrt(256 + num^2)) end if
    end do
end proc
> pyth( )
                                12, 16, 20
                                16, 30, 34
                                16, 63, 65                         (4)
>
> #version 2
> pyth2 := proc( )
    local num;
    for num from 1 to 1000 do
    if frac(sqrt(16^2 + num^2)) = 0 then print(sort([num, 16, sqrt(16^2 + num^2)]));
    end if;
    end do;

    end proc;
pyth2 := proc( )                                                   (5)
    local num;
    for num to 1000 do
        if frac(sqrt(256 + num^2)) = 0 then print(sort([num, 16, sqrt(256 + num^2)]))
        end if
    end do
end proc
> pyth2( );
                              [12, 16, 20]
                              [16, 30, 34]
                              [16, 63, 65]                         (6)
>
>
> #Q3
> q3proc := proc(L :: list)
    local num;
    for num from 1 to nops(L) do
    if type(L[num], integer) = false then return false;
    end if;
    end do;
    return true;
```

```
    end proc;
q3proc := proc(L::list)                                                    (7)
    local num;
    for num to nops(L) do if type(L[num], integer) = false then return false end if end do;
    return true
end proc
>
>
> L1 := [1, 2, 3, 3, 3, 4, 6, 7, 8];
                          L1 := [1, 2, 3, 3, 3, 4, 6, 7, 8]                (8)
> L2 := [1.1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 3];
                       L2 := [1.1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 3]           (9)
> q3proc(L2);
                                  false                                    (10)
> q3proc(L1);
                                  true                                     (11)
>
>
> #Q4
> q4proc :=proc(L :: list, k :: posint)
    local num, mymin, minindex;
    if k > nops(L) then print("Error: k is too large"); return 0;
    else
     mymin := L[1];
     minindex := 1;
     for num from 2 to k do
     if mymin > L[num] then mymin := L[num]; minindex := num;
     end if;
     end do;
     return minindex;
    end if;
    end proc;
q4proc := proc(L::list, k::posint)                                         (12)
    local num, mymin, minindex;
    if nops(L) < k then
        print("Error: k is too large"); return 0
    else
        mymin := L[1];
        minindex := 1;
        for num from 2 to k do
            if L[num] < mymin then mymin := L[num]; minindex := num end if
        end do;
        return minindex
    end if
end proc
```

```
> L := [12, 43, 52, 8, 1]
```

$$L := [12, 43, 52, 8, 1] \quad \textbf{(13)}$$

```
> q4proc(L, 9);
```

"Error: k is too large"

$$0 \quad \textbf{(14)}$$

```
> q4proc(L, 3);
```

$$1 \quad \textbf{(15)}$$

```
> q4proc(L, 4);
```

$$4 \quad \textbf{(16)}$$

```
>
```