Q1.

1- For n>2 , we have to show that f(n) = f(n-1) + f(n-2):

Standing on ground, we want to count the number of ways we can reach the nth stair. Before we reach nth stair, we will reach (n-1)th stair or (n-2)th stair, and from that we have just one option in either case i.e. to climb: 1 stair or 2 stairs respectively. So we can count the number of ways we will reach (n-1)th stair or (n-2)th stair.

Therefore, we can say that the number of ways to reach nth stair is same as the addition of the ways to reach (n-1)th stair and (n-2)th stair.

Therefore f(n) = (n-1) + f(n-2)

For n = 0 we don't need to count as we are not climbing.

For n =1 we can easily tell that we have just a single option to reach 1st stair; that is by taking 1 step from ground.

For n =2 we can tell that we have two options to take 1 stair twice or to take 2 stairs once.

Therefore f(0) = 0 , f(1) =1 and f(2) = 2;

And for n>2 we have, f(n) = (n-1) + f(n-2)
Hence, proved.

2- I have defined an algorithm which has a function (climbStair(n)) and that is used in computation of the recursive function [f(n) = (n-1) + f(n-2)] for all n>2 and for the values of n = 0, 1, 2 ; I have specified the values as 0, 1, 2 respectively.

For n < 0, I have raised an exception "Invalid stair count".

3- To prove : f(n) = 2 + $\sum_{i=1}^{n-2} f(i)$

Let's prove this using Induction:

Base Case:

n = 3

f(3) = total cases = f(1) + f(2) = 1+2 = 3; using: { f(n) = (n-1) + f(n-2) }

f(3) = 2+ $\sum_{i=1}^{1} f(i)$ = 2 + 1

Induction Hypothesis:

f(m) is true => f(m) = 2 + $\sum_{i=1}^{m-2} f(i)$ for all m<=n

for some m, n∈ N .

Induction Step:

To prove, f(n+1) = 2 + $\sum_{i=1}^{n-1} f(i)$

f(n+1) = f(n) + f(n-1)

f(n) = 2 + $\sum_{i=1}^{n-2} f(i)$ and f(n-1) = 2 + $\sum_{i=1}^{n-3} f(i)$

because {n, n-1} <=n

f(n) + f(n-1) = 2 + $\sum_{i=1}^{n-2} f(i)$ + f(n-1) = 2 + $\sum_{i=1}^{n-3} f(i)$

= 4 + f(1) + f(2) +......... + f(n-2) +

+ f(1) + f(2) +.... + f(n-3)

Now we can group {f(1) +f(2)}, { f(2) +f(3)},......,{f(n-3) +f(n-2)}

as f(3),f(4),........,f(n-1) ; using: { f(n) = (n-1) + f(n-2) }

so we can write

f(n+1)= f(n) + f(n-1) = 4 – f(2) +f(1) +f(2) +f(3)+..........+f(n-1)

which is same as writing

f(n+1) = 2 + $\sum_{i=1}^{n-1} f(i)$

Hence, proved.

Q2.

1- To define a recursive relation in f(n) and [f(n/10)] { [] denotes floor of n}, according to the function given in the question f(n) = $2^k d_k$+..........+$2^0 d_0$ where n be a positive integer with digits d_k,........, d_{k1}, d_0 , d_k being the most significant digit,
   We extract the digits from the number ‘n’ by using a recursive relation in f(n) and [f(n/10)]

i- We first find the remainder/(least significant digit) of n when divided by 10

ii- We then define a recursive relation in f(n) and [f(n/10)] as:

$$f(n) = \begin{cases} 0 & if\ n = 0 \\ (n) - \left[\left(\frac{n}{10}\right)\right] + 2[f(n/10)] & else \end{cases}$$

We are doing a recursive call in which we are changing the value of n each time the least significant digit is extracted to $[\mathrm{f}(\mathrm{n}/10)]$, so that next time we can extract the next least significant digit.

2- The function modifiedDigitSum (n) is defined as int → int to follow the above procedure to determine the value of f(n) for any n.

   The function modifiedDigitSum(n) gets called each time after a digit is extracted and the process continues till we get the final value of n= 0 and the desired sum is obtained.

Q3.

1- To find the number of positive integers less than or equal to n, that are expressible as sum of squares of two (not necessarily distinct) natural numbers a and b, we design the following algorithm:

We firstly define a function sqrt(n): int → int, which is used to compute the square root of perfect squares only.

To design it we define a local function g(n): int → int, that uses recursion to find if the number is a perfect square.

$$g(n): int \rightarrow int \;=\; \begin{cases} if\ n < 0\ then\ raise\ exception\ ("Invalid\ input") \\ \quad else\ if\ n\ =\ a^2 then\ return\ a \\ \quad else\ if\ n <\ a^2\ then\ return\ 0 \\ \quad\quad else \quad g(a+1) \end{cases}$$

This algorithm checks if any integer n is a perfect square and if it is; this prints its square root.

<u>Basis: We are finding any integer b such that a^2 + b^2 <= n ; for some known n and any varying number 'a' <= $\sqrt{n}$</u>

The next function we define squaredCount(n): int→ int to find the number of such integers (a, b) for any given n .

So we define another local function local to squaredCount(n) -> q(n) as below:

In the local function we define another local variable: b = sqrt(n-a^2) which computes only when (n-(a*a)) is a perfect number.

$$q(n,a) : int*int \rightarrow int \ = \begin{cases} if\ n < 0\ then\ return\ 0 \\ else\ \ let\ b\ =\ sqrt(n - a^2)\{ \\ if\ b > 0\ then\ return\ 1 \\ else\ q(n, a+1) \end{cases}$$

After this step we use a recursion in squaredCount(n) to compute for values less than n. As such:

$$squaredCount(n) = \begin{cases} if\ n = 0\ then\ return\ 0 \\ else\ q(n,0) + sumOfSquares(n-1) \end{cases}$$

This results in the total of all numbers that are expressible as sum of squares of two natural numbers a and b.

Ex: for squaredCount(8) we have

$8 = 2^2 + 2^2 \rightarrow +1$

$7 \rightarrow +0$

$6 \rightarrow +0$

$5 = 1^2 + 2^2 \rightarrow +1$

$4 = 0^2 + 2^2 \rightarrow +1$

$3 \rightarrow +0$

$2 = 1^2 + 1^2 \rightarrow +1$

$1 = 1^2 + 1^2 \rightarrow +1$

Total numbers = 5;

Hence, proved.

Q4.

1- The function is written in the sml file.
2- I have designed an algorithm to compute the sum of π. It goes as follows:
   a- A function nilakanthaSum(t): (real → real) is used as defined to calculate the sum.
   b- A local variable 'N' is defined which gives the greatest integer <=t. I have used floor operator to get the greatest integer <=t. The floor operator rounds off the real number t to an integer N which is the greatest integer less than/equal to t.
   c- The nilakanthaSum function is now defined as
   If N is even then we have a specific condition that we want that
   (-1)* nilakanthaSum(t)

$$\textit{nilakanthaSum(t)} = \begin{cases} if\ N = 0\ then\ return\ 3.0 \\ else\ if\ N\ mod\ 2 = 1\ then\ nilakanthaSum(t - 1.0) \\ \quad + \frac{4.0}{(2.0*t)*(2.0*t\ +1.0)*(2.0*t\ +2.0)} \\ else\ nilakanthaSum(t - 1.0) \\ \quad - \frac{4.0}{(2.0*t)*(2.0*t\ +1.0)*(2.0*t\ +2.0)} \end{cases}$$

   d- Now as we know that
   For the
   Base Case: t =1; We have the output as 3 + $\frac{4}{2*3*4}$ = 3.166 and $\left\{\frac{4}{2*3*4}\right\}$ <1 ; Hence base case is true.
   Induction Hypothesis:
   Assuming for all k<t, this algorithm is true. For all k, t ∈ N.
   Induction Step:
   For n, we have 4/(2*t)*(2*t + 1)*(2*t + 2) which is < 1 for all n>1.
   Hence we can say that induction step is also true.

Hence proved.