Sequences - Algorithms

Proofreading of English by Laurence Weinstock

Contents

1 Sequences : general overview

1.1 Definition

Definition 1 : A sequence (u_n) is a function whose domain is $\mathbb{N}$ (or possibly $\mathbb{N} - [\![0, k]\!]$) and whose codomain is $\mathbb{R}$, that relates a real number written u_n to an integer n.

$$\begin{aligned}(u_n) \quad : \quad \mathbb{N} \text{ or } \mathbb{N} - [\![0,k]\!] &\longrightarrow \mathbb{R}\\ n &\longmapsto u_n\end{aligned}$$

Note :

- $\mathbb{N} - [\![0, k]\!]$ is the set $\mathbb{N}$ minus the first k natural numbers.
- u_n is called the general term of the sequence (u_n).
- Note the difference between the sequence (u_n) and the general term u_n
- If a sequence is defined from the index p onwards, it is denoted $(u_n)_{n\geqslant p}$

Examples :

- (u_n) : 2 ; 5 ; 8 ; 11 ; 14 ; 17 ; . . . an arithmetic sequence
- (v_n) : 3 ; 6 ; 12 ; 24 ; 48 ; 96 ; . . . a geometric sequence

1.2 Examples of sequences

a) **Explicitly defined sequences** : $u_n = f(n)$:

$$u_n = \frac{1}{n}, \quad n \in \mathbb{N}^*, \qquad v_n = \sqrt{n-3}, \quad n \geqslant 3$$

b) **Recursive sequences** defined by a recurrence relation and one or more starting values :

- By one starting value : $u_{n+1} = f(u_n)$

$$\begin{cases} u_0 = 4 \\ u_{n+1} = 0.75u_n + 2 \end{cases}$$

To calculate u_n, given n

```
Variables: $N, I$ integers
           $U$ real number
Inputs and initialization
    Read $N$
    $4 \to U$                         returns $u_0$
Processing
    for $I$ from 1 to $N$ do
        $0,75U + 2 \to U$             recurrence
                                      relation
    end
Output   : Print $U$
```

N	5	10	20	30
U	7.050 8	7.774 7	7.987 3	7.999 9

The sequence seems to be increasing and to converge to 8

- By two starting values : $u_{n+2} = f(u_{n+1}\,,\, u_n)$

$$\begin{cases} u_0 = 1, \quad u_1 = 1 \\ u_{n+2} = u_{n+1} + u_n \end{cases}$$

To calculate u_n, given n

```
Variables: $N, I$ integers
           $U, V, W$ real numbers
Inputs and initialization
    Read $N$
    $1 \to V$          returns $u_0$
    $1 \to U$          returns $u_1$
Processing
    for $I$ from 2 to $N$ do
        $U + V \to W$          recurrence relation
        $V \to U$  }  the next term in the
        $W \to U$  }  sequence is calculated
    end
Output   : Print $U$
```

N	10	15	20	30
U	89	987	10 946	1 346 269

c) A sequence can be defined via another sequence or a sum of terms, ...

Given (u_n) , the sequence (v_n) can be defined by : $v_n = u_n - 4$

$$w_n = \sum_{i=1}^{n} \frac{1}{i} = 1 + \frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{n}$$

To determine an approximate value of a particular term (w_n), the following program can be written :

```
Variables: $N, I$   integers
           $W$ real number
Inputs and initialization
    Read $N$
    $0 \to W$
Processing
    for $I$ going from 1 to $N$ do
        $W + \frac{1}{I} \to W$
    end
Output   : Print $W$
```

For instance, to find the values for w_5, w_{10}, w_{50}.

If one wants to find the exact result as a fraction with the TI 82, type : "Disp W ▷Frac"

We find the following values :

- $w_5 = \dfrac{137}{60} \simeq 2.283$
- $w_{10} \simeq 2.923$, $w_{50} \simeq 4.499$

d) A sequence can also be defined by an explicit formula without specifying the value of any term.

For instance, the sequence (d_n) that relates the nth term d_n to the nth decimal of the number $\pi = 3{,}141\,592\ldots$: $d_1 = 1$, $d_2 = 4$, $d_3 = 1$, $d_4 = 5$, $d_5 = 9$, $d_6 = 2\ldots$

1.3 Monotonicity of a sequence

Definition 2 : Let (u_n) be a numeric sequence. The sequence (u_n) is said to be :

- strictly **increasing** (from index k) if
$$u_{n+1} > u_n \quad \text{for all integers} \quad n \geqslant k$$
- strictly **decreasing** (from index k) if
$$u_{n+1} < u_n \quad \text{for all integers} \quad n \geqslant k$$
- **monotonic** (from index k) if it is increasing or decreasing from index k
- **stationary** if there is an index k such that
$$u_{n+1} = u_n \quad \text{for all integers} \quad n \geqslant k$$
- **constant** if $u_{n+1} = u_n$ for all integers n.

Note :

There are sequences that are neither increasing nor decreasing : $a_n = (-1)^n$
The first terms of the sequence are not necessarily important for determining the behavior of a sequence. They can, however, give an indication as to the monotonicity of the sequence.

1.4 Showing that a sequence is monotonic

Law 1 : In order to show that a sequence is monotonic :

- Determine the sign of the difference $u_{n+1} - u_n$
  If the difference is positive (resp. negative) from index k onwards, then the sequence is increasing (resp. decreasing) for $n \geqslant k$
- If all the terms of the sequence are strictly positive from index k onwards, then we can compare the ratio $\dfrac{u_{n+1}}{u_n}$ to 1
  If the ratio is greater than 1 (resp less than 1) from index k onwards, the sequence is increasing (resp. decreasing) for $n \geqslant k$
- If the sequence is defined explicitly, then we can determine whether the function f is monotonic in $\mathbb{R}_+$
- Using mathematical induction (see next chapter).

Examples :

- Show that the sequence (u_n) defined for all n by : $u_n = n^2 - n$ is increasing.
  First, determine the difference : $u_{n+1} - u_n$

$$\begin{aligned} u_{n+1} - u_n &= (n+1)^2 - (n+1) - (n^2 - n) \\ &= n^2 + 2n + 1 - n - 1 - n^2 + n \\ &= 2n \end{aligned}$$

  Seeing as $2n \geqslant 0$ for all $n \in \mathbb{N}$, we have $u_{n+1} - u_n \geqslant 0$. The sequence (u_n) is increasing from index 0.

- Show that the sequence (u_n) defined for all $n \in \mathbb{N}^*$ by : $u_n = \dfrac{2^n}{n}$ is increasing.

  Given that for all $n \in \mathbb{N}^*$, $u_n > 0$, it is possible to compare the ratio $\dfrac{u_{n+1}}{u_n}$ to 1 :

$$\frac{u_{n+1}}{u_n} = \frac{\dfrac{2^{n+1}}{n+1}}{\dfrac{2^n}{n}} = \frac{2^{n+1}}{n+1} \times \frac{n}{2^n} = \frac{2n}{n+1}$$

  Seeing as $n \geqslant 1$, by adding n to each side of the inequality, we find that $2n \geqslant n+1$, hence :

$$\frac{2n}{n+1} \geqslant 1$$

  As $\forall n \geqslant 1 \quad \dfrac{u_{n+1}}{u_n} \geqslant 1$, the sequence (u_n) is increasing from index 1.

- Show that the sequence (u_n) defined for all $n \geqslant 2$ by : $u_n = \dfrac{2n+1}{n-1}$ is decreasing.

  Let us determine whether the function f defined by $f(x) = \dfrac{2x+1}{x-1}$ and whose domain is $I = [2\,;+\infty[$ is monotonic. This function is differentiable on I, so

$$f'(x) = \frac{2(x-1) - (2x+1)}{(x-1)^2} = \frac{-3}{(x-1)^2} \quad \text{therefore} \quad f'(x) < 0 \quad x \in I$$

  The function f is decreasing on I, so the sequence (u_n) is decreasing.

1.5 Graphing a sequence

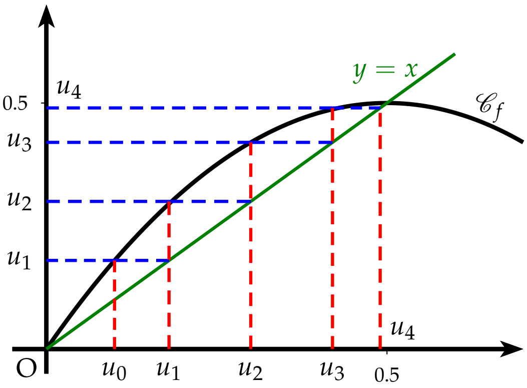
To visualize a sequence defined by the recurrence relation $u_{n+1} = f(u_n)$, we can plot the graph of the function f and the line with the equation $y = x$. The line is used to carry over the terms of the sequence onto the x-axis.

Consider the sequence (u_n) defined by :

$$\begin{cases} u_0 = 0.1 \\ u_{n+1} = 2u_n(1-u_n) \end{cases}$$

We obtain the following representation of the sequence after plotting the graph $\mathscr{C}_f$ of the function f defined by :

$$f(x) = 2x(1-x)$$

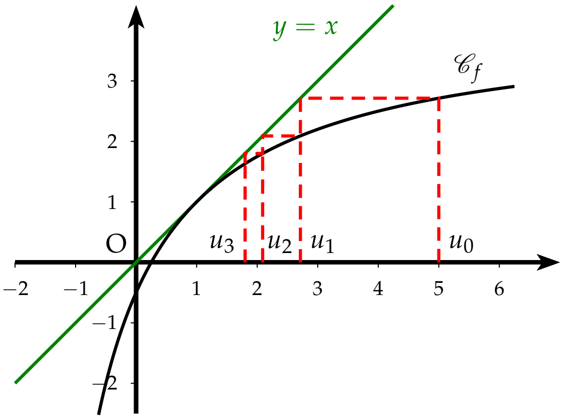


Example : Consider the sequence (u_n) defined by :

$$\begin{cases} u_0 = 5 \\ u_{n+1} = \dfrac{4u_n - 1}{u_n + 2} \quad \forall n \in \mathbb{N} \end{cases}$$

f is the function defined on the interval $]-2\,;\,+\infty[$ by $f(x) = \dfrac{4x-1}{x+2}$.

After plotting the graph $\mathscr{C}_f$ of the function f and the line with the equation $y = x$, place u_0 on the x-axis, then construct u_1, u_2 and u_3 leaving the lines traced on the graph. What can be conjectured with regards to the monotonicity and convergence of the sequence (u_n) ?



Using this graph, we can conjecture that :

- the sequence is decreasing
- the sequence seems to converge to 1 which is the x-coordinate of the point of intersection between the line and the graph of the function.

To graph a sequence with the TI 82, select "Seq" mode and format "web". Input the sequence then press "graph" and "trace".

Input the sequence after pressing "$f(x)$"

$n_{min} = 0$

$u(n) = (4u(n-1) - 1)/(u(n-1) + 2)$

$u(n_{min}) = 5$

For the viewing window :

$n_{min} = 0$, $n_{max} = 3$,

PlotStart = 1, PlotStep = 1,

$X_{min} = -2$, $X_{max} = 6$, $X_{scl} = 1$

$Y_{min} = -2$, $Y_{max} = 4$, $Y_{scl} = 1$

2 Arithmetic sequences (review)

2.1 Definition

Definition 3 : An arithmetic sequence (u_n) is defined by :

- a first term u_0 or u_p
- a recurrence relation : $u_{n+1} = u_n + r$
  r ("raison" in French) common difference in English

Note : An arithmetic sequence is the same thing as an arithmetic progression.

Example : $(u_n) : \begin{cases} u_0 = 2 \\ u_{n+1} = u_n + 3 \end{cases}$ $\qquad (u_n) : 2 ; 5 ; 8 ; 11 ; 14 ; 17 ; \dots$

2.2 How to recognize an arithmetic sequence

A sequence (u_n) is arithmetic if the difference between consecutive terms is constant. This constant is the common difference ("raison" in French).

$$\forall n \geqslant p \qquad u_{n+1} - u_n = r \quad \Leftrightarrow \quad (u_n) \text{ is arithmetic with a common difference of } r$$

Example : Given the sequence (u_n) defined by $u_n = 4n - 1$, show that the sequence (u_n) is arithmetic.

$\forall n \in \mathbb{N}$, $u_{n+1} - u_n = 4(n+1) - 1 - (4n - 1) = 4$. (u_n) is an arithmetic sequence.

2.3 Expression of the general term as a function of n

Law 2 : Let (u_n) be an arithmetic sequence with a common difference of r

- If the first term is u_0, then : $u_n = u_0 + nr$
- If the first term is u_p, then : $u_n = u_p + (n - p)r$

2.4 Sum of the first n terms : finite series

Theorem 1 : The sum of the first terms of an arithmetic sequence is :

$$S_n = \text{Number of terms} \times \frac{\Sigma \text{ first and last terms}}{2}$$

$$S_n = 1 + 2 + 3 + \cdots + n \quad \text{then} \qquad S_n = \frac{n(n+1)}{2}$$

$$S_n = u_0 + u_1 + u_2 + \cdots + u_n \quad \text{then} \qquad S_n = (n+1)\left(\frac{u_0 + u_n}{2}\right)$$

Example : Calculate the sum of the following terms :

$$S = 8 + 13 + 18 + \cdots + 2008 + 2013$$

This is the sum of the terms of an arithmetic sequence with a common difference of 5, the first and last terms being 8 and 2013.

There are : $\dfrac{2013-8}{5} + 1 = 402$ terms.

$$S = 402 \times \frac{8 + 2013}{2} = 406\,221$$

Algorithm : Verification

The following algorithm is written to calculate the sum S.

As the arithmetic sequence is increasing, we can use the conditional loop as shown opposite. Note that the condition is $u \leqslant 2008$ instead of $u \leqslant 2013$ because the last loop with this condition calculates the next term 2013 which is added to the sum S.

The program then returns the calculated result.

```
Variables: $I$ integer
           $U, S$ real numbers
Inputs and initialization
    $8 \to U$
    $8 \to S$
Processing
    while $U \leqslant 2008$ do
        $U + 5 \to U$
        $S + U \to S$
    end
Output  : Print $S$
```

3 Geometric sequences (review)

3.1 Definition

Definition 4 : A geometric sequence (u_n) is defined by :

- a first term u_0 or u_p
- a recurrence relation : $u_{n+1} = q \times u_n$
  q ("raison" in French) common ratio in English

Note : A geometric sequence is the same thing as a geometric progression.

Example : $(u_n) : \begin{cases} u_0 = 3 \\ u_{n+1} = 2u_n \end{cases}$ $\qquad (u_n) : 3 ; 6 ; 12 ; 24 ; 48 ; 96 ; \ldots$

3.2 How to recognize a geometric sequence

A sequence is geometric if the quotient (ratio) of consecutive terms is constant. This constant is the common ratio.

$$\forall n \geqslant p \qquad \frac{u_{n+1}}{u_n} = q \quad \Rightarrow \quad (u_n) \text{ is geometric with a common ratio of } q$$

Example : Consider the sequence $(u_n) : \begin{cases} u_0 = 5 \\ u_{n+1} = 3u_n - 2 \end{cases}$

Let $v_n = u_n - 1$. Show that the sequence (v_n) is geometric.

$$v_{n+1} = u_{n+1} - 1 = 3u_n - 2 - 1 = 3(u_n - 1) = 3v_n \quad \text{so } \forall n \in \mathbb{N}, \ \frac{v_{n+1}}{v_n} = 3$$

The sequence (v_n) is geometric with a common ratio of 3 and a first term of $v_0 = 4$.

3.3 Expression of the general term as a function of n

Law 3 : Let (u_n) be a geometric sequence with a common ratio q

- If the first term is u_0, then : $u_n = q^n \times u_0$
- If the first term is u_p, then : $u_n = q^{n-p} \times u_p$

3.4 Sum of the first n terms : finite series

Theorem 2 : The sum of the first n terms of a geometric sequence ($q \neq 1$) is :

$$S_n = \text{1st term} \times \frac{1 - q^{\text{number of terms}}}{1 - q}$$

$$S_n = u_0 + u_1 + u_2 + \cdots + u_n \quad \text{then} \quad S_n = u_0 \times \frac{1 - q^{n+1}}{1 - q}$$

Example : Calculate the sum of the following terms :

$$S = 0.02 - 0.1 + 0.5 - 2.5 + \cdots + 312.5$$

This is the sum of the terms of a geometric sequence (u_n) with a common ratio of $q = -5$ and a first term of $u_0 = 0.02$.

We must have : $u_n = 0.02(-5)^n = 312.5 \Rightarrow (-5)^n = \dfrac{312.5}{0.02} = 15\,625 = 5^6$.

There are 7 terms. $S = 0.02 \times \dfrac{1 - (-5)^7}{1 - (-5)} = \dfrac{5^7 + 1}{300} = 260.42$

3.5 Limit of a geometric sequence

Theorem 3 : Consider a sequence (u_n) defined by : $u_n = q^n$ (with $q \in \mathbb{R}$)

- If $q > 1$ then (u_n) is divergent and $\lim\limits_{n \to +\infty} q^n = +\infty$
- If $q = 1$ then (u_n) is constant (thus converging to 1)
- If $-1 < q < 1$ then (u_n) is convergent and $\lim\limits_{n \to +\infty} q^n = 0$
- If $q \leqslant -1$ then (u_n) is divergent and has no limit

Examples :

- Let (u_n) be a geometric sequence : $u_0 = -2$ and $q = 1.5$. Does the sequence (u_n) converge ?

  $\lim\limits_{n \to +\infty} 1.5^n = +\infty$ because $1.5 > 1$. The sequence (u_n) diverges to $+\infty$.

- Let (v_n) be a geometric sequence : $v_0 = 4$ and $q = \dfrac{3}{4}$. Does the sequence (v_n) converge ?

  $\lim\limits_{n\to+\infty} \left(\dfrac{3}{4}\right)^n = 0$ because $-1 < \dfrac{3}{4} < 1$. The sequence (v_n) converges to 0.

4 Algorithms

4.1 Introduction

Definition 5 : An algorithm is a series of instructions, which when executed correctly, leads to an outcome.
In order to run smoothly, an algorithm must only contain instructions understandable by those who must execute it.

Two qualities are needed to design an algorithm :

- **Intuition**, because *there is no pre-defined procedure that allows us to know in advance which instructions will lead to the desired result*.
- **Methodology and rigor**. As a matter of fact, every time a series of instructions is written which is believed to be correct, it is always necessary run it through the computer to check whether the result is the correct one.

All computers are inherently capable of understanding only four categories of orders (in programming, the word *instructions* is used rather that the word *orders*). These four categories of instructions are :

- Assigning a value to a variable
- Reading / writing
- Tests
- Loops

With these instructions, an algorithm solves a problem independently of the particularities of a programming language .

Learning to write an algorithm is learning to manage the logical structure of a computer program.

"*A programming language is a convention to give orders to a computer*."(Pascal, Fortran, C ++, PHP, Mathematica . . .)

4.2 Writing conventions for algorithms

Historically, algorithms were represented in many different ways.

- They were especially represented as diagrams called flowcharts, with squares, rhombuses, etc. However as soon as the algorithm begins to grow a little, this approach becomes impractical. .
- Nowadays, a series of writing conventions called "pseudocode" is used. Pseudocode looks like a real programming language but is unhindered by the syntax problems of a programming language. Pseudocode is likely to vary somewhat from manual to manual (or from one teacher to the next) : no computer is supposed to recognize it.

4.3 Variables

4.3.1 Definition

Definition 6 : As soon as one needs to store information in a program, a variable is used. A variable can be thought of as a box, that the program (the computer) recognizes by its label. The box is labeled so that its contents can be accessed.

4.3.2 Variable declaration

The first thing to do before you can use a variable is to create a box and give it a label. This is called variable declaration.

The name of the variable (the label of the box) must follow strict rules that vary according to the programming language. However, an iron clad rule is that a variable can be named with letters and numbers, but most signs, punctuation, and particularly spaces, are excluded. A correct variable name must start with a letter. The maximum number of characters that can be used depends on the programming language.

Once the name has been chosen, the type of the variable must be determined. Variables fall into three main categories (there are others !) :

- Alphanumeric : for text.
- Numeric : for a number (integer, decimal, real number).
- Lists : for ordered sets of numbers.

4.4 Assigning a numeric value to a variable

The only thing that can be done with a variable is to assign a value to it.

$\boxed{a \leftarrow 24}$	Assign the value 24 to the variable a	$\boxed{24 \rightarrow a}$
$\boxed{a \leftarrow b}$	Assign the value b to the variable a	$\boxed{b \rightarrow a}$

with an operation :

$\boxed{c \leftarrow a+4}$	Assign the value $a+4$ to the variable c	$\boxed{a+4 \rightarrow a}$

with an increment :

$\boxed{b \leftarrow b+1}$	Increment the value of the variable b by 1	$\boxed{b+1 \rightarrow b}$

The numerical operators are :

- Addition $+$
- Subtraction $-$
- Multiplication $*$
- Division $/$
- Exponentiation $\hat{}$

The logical operators are :

- AND intersection of two sets
- OR union of two sets
- NOT complementary to a set

⚠ Note :

- A variable can only have one value at a time
- Assigning a variable is often written : $B = A$ or $B := A$ which means that B takes the value of A, which is different from $A = B$ or $A := B$ where A takes the value of B.

4.5 Reading and writing a variable

Definition 7 : To **Read** or **Input** a variable means that the user must enter a value for the program to read it.

To **Write** or **Print** a variable means that the program returns the value of the variable that the program has computed.

4.6 Tests

There are two forms of testing : full form or simplified form :

Full form	**Simplified form**
If condition **then**	**If** condition **then**
Instructions 1	Instructions
else	**EndIf**
Instructions 2	If the condition is not satisfied, the program does nothing.
EndIf	

The condition can be :

- A target value.
- A comparison with another variable (equality, inequality, not-equality).

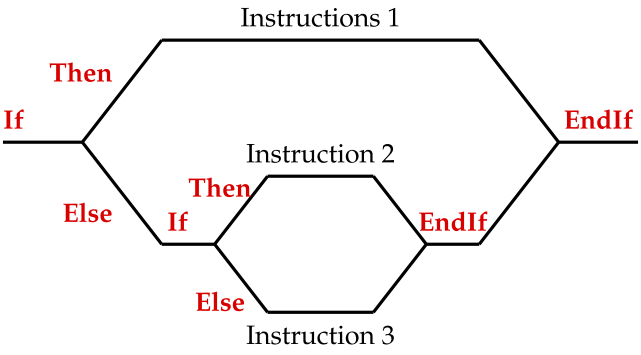
It is also possible to create a test with several conditions connected by a logical operator :

- condition 1 AND condition 2 : both conditions must be satisfied at the same time.
- condition 1 OR condition 2 : at least one of the two conditions must be satisfied.

Another test can also be used within the first test :

```
If   condition 1   then
     Instructions 1
else          If   condition 2   then
                        Instructions 2
                   else
                        Instruction 3
                   EndIf
EndIf
```

This could be simplified as follows : :



For example, solving quadratic equations :

$$Ax^2 + Bx + C = 0$$

The program shown opposite is designed to solve quadratic equations.

Note that the first simplified "If" test determines whether or not the equation is of the first degree. If this condition is satisfied then the program terminates.

There are then two "If" tests for the three cases of the variable Δ.

Note that quotation marks must be used in order to print a text.

```
Variables: $A, B, C, X, Y, \Delta$ real numbers
Inputs and initialization
    Input   $A, B, C$
    $B^2 - 4AC \rightarrow \Delta$
Processing and outputs
    if $A = 0$ then
        $-\frac{C}{B} \rightarrow X$
        Print $X$
        Go to the end
    end
    if $\Delta > 0$ then
        $\frac{-B+\sqrt{\Delta}}{2A} \rightarrow X$
        $\frac{-B-\sqrt{\Delta}}{2A} \rightarrow Y$
        Print $X, Y$
    else
        if $\Delta = 0$ then
            $-\frac{B}{2A} \rightarrow X$
            Print $X$
        else
            Print "No solution"
        end
    end
```

4.7 Loops

4.7.1 Definition

Definition 8 : A loop is a repetitive or iterative structure, i.e. the loop performs n times a set of instructions under the control of a stop condition.

4.7.2 The while loop

The while loop adheres to the following format :

```
While   condition
     Instructions
EndWhile
```

⚠ If the condition is always satisfied, the computer runs the loop forever. The "infinite loop" is one of the biggest fears of any programmer.
This programming error is commonly made by programmer trainees .

Example :

Find the integer N such that :

$$1 + 2 + \cdots + N > 10^P$$

This is a loop controlled by the condition $U < 10^P$ that calculates the sum of the N first natural numbers by incrementing N by one for each turn of the loop.

- If $P=3$, the algorithm returns $N=45$
- If $P = 6$, the algorithm returns $N = 1414$

```
Variables: $N, P$ integers
           $U$ real number
Inputs and initialization
    Input $P$
    $0 \to N$
    $0 \to U$
Processing
    while $U \leqslant 10^P$ do
        $N + 1 \to N$
        $U + N \to U$
    end
Output : Print $N$
```

4.7.3 Loop counter

If the number of increments is known in advance, then the following structure is used :

```
For counter = initial value to final value from step value
    Instructions
EndFor
```

Example :

Let us create a program to calculate K ! "factorial K" :
$N = K! = 1 \times 2 \times 3 \times \cdots \times K$
The counter I is incremented from 2 to K (The step is equal to 1 by default).
We then obtain $N = K!$

- If $K = 8$, the algorithm returns $N = 40\,320$
- If $K = 12$, the algorithm returns : $N = 479\,001\,600$

```
Variables: $K, I, N$ integers
Inputs and initialization
    Read $K$
    $1 \to N$
Processing
    for $I$ from 1 to $K$ do
        $N \times I \to N$
    end
Output : Print $N$
```