|  |  |  |
| --- | --- | --- |
| **ACTIES OP HET SCHERM** | **VOICE-OVER** | **DUUR** |
|  | Arrays are a very important data structure in scientific computing. In Fortran, arrays are first-class citizens since array arithmetic is supported directly in the core language. One-dimensional arrays can represent vectors, two-dimensional arrays matrices and so on. |  |
| 1. Add array declaration for coords 2. Assign 5.7 to coords(2) 3. Assign 2\*coords(2) to coords(1) 4. Add print statement 5. Save file | 1. We can declare an array variable by adding the attribute “dimension” to its declaration. For instance, this declaration means that we have an array of type “real” with 3 elements. This reserve memory for 3 floating point values. The name of the arrays is “cords”. 2. Individual array elements can be assigned to by using an index. Fortran’s array are by default 1-based indexed, i.e., the first element has index 1, the second index 2 and so on. The last element has the length of the array as index. Here, we assign the value 5.7 to the second element of “coords”. 3. Using the value of an array element works similarly of course, here we assign 2 times the value of the second element to the first element of the array. 4. It is easy to print all elements of an array by simply using the asterisk format specifier. |  |
| 1. Switch to terminal 2. Compile code 3. Run application | 1. When we compile and run the code, we get the expected output for the first two array elements, but what about the third element? Since the latter wasn’t initialized, it will contain whatever happens to be at the corresponding memory location, so essentially a random number. Very often this is zero, or at least very close to zero, but that is not necessarily the case. |  |
| 1. Add coords initialization to 1.0 2. Add print of 3.0 + coords 3. Add print of 2.5\*coords | 1. It is quite easy to initialize all elements of an array to some value, for instance 1.0. Assigning a scalar value to an array means that this scalar value will be assigned to all array elements. 2. Adding a scalar to an array implies adding that scalar to each array element. 3. Likewise multiplying a scalar and an array results in the element-wise multiplication. |  |
| 1. Swith to terminal 2. Compile code 3. Run application | 1. When we compile the code and run the application, we get the expected results. |  |
| 1. Add print of coords\*\*2 2. Add print of dot\_product | 1. Also the power operator works element-wise, which may be unexpected. The dot product of two vectors can be computed using the “dot\_product” intrinsic function. |  |
| 1. Switch to terminal 2. Compile source code 3. Run application | 1. Again, compiling and running shows the expected output |  |
| 1. Add array coords1 and coords2, initialize with implied do 2. Add print of coords1 + coords2, print of coords1\*coords2, print of dot\_product 3. Add print of size and sum | 1. Arrays can also be initialized by a sequence of values or even an implied do loop. This array has three elements, the first will have the value -0.5, the second 0.0 and the third 0.5. 2. We can now add the two arrays element-wise, or multiply them element-wise, and again, the dot product. 3. Many intrinsic functions operate on arrays, for instance “size” and “sum”. |  |
| 1. Switch to terminal 2. Compile 3. Run | 1. Again, when we run this application, we get the results we expect. |  |
|  |  |  |
| **TOTALE DUUR** | | *Maak je screencast niet langer dan ca. 6 minuten.* |