WIKIPEDIA
The Free Encyclopedia

Iteration

Iteration is the repetition of a process in order to generate a (possibly unbounded) sequence of outcomes. Each repetition of the process is a single iteration, and the outcome of each iteration is then the starting point of the next iteration.

In mathematics and computer science, iteration (along with the related technique of recursion) is a standard element of algorithms.

Mathematics

In mathematics, iteration may refer to the process of iterating a function, i.e. applying a function repeatedly, using the output from one iteration as the input to the next. Iteration of apparently simple functions can produce complex behaviors and difficult problems – for examples, see the Collatz conjecture and juggler sequences.

Another use of iteration in mathematics is in iterative methods which are used to produce approximate numerical solutions to certain mathematical problems. Newton's method is an example of an iterative method. Manual calculation of a number's square root is a common use and a well-known example.

Computing

In computing, iteration is the technique marking out of a block of statements within a computer program for a defined number of repetitions. That block of statements is said to be *iterated*; a computer scientist might also refer to that block of statements as *an* "iteration".

Implementations

Loops constitute the most common language constructs for performing iterations. The following pseudocode "iterates" three times the line of code between begin & end through a *for loop*, and uses the values of *i* as increments.

```
a := 0
for i := 1 to 3 do       { loop three times }
begin
    a := a + i;          { add the current value of i to a }
end;
print(a);                { the number 6 is printed (0 + 1; 1 + 2; 3 + 3) }
```

It is permissible, and often necessary, to use values from other parts of the program outside the bracketed block of statements, to perform the desired function.

Iterators constitute alternative language constructs to loops, which ensure consistent iterations over specific data structures. They can eventually save time and effort in later coding attempts. In particular, an iterator allows one to repeat the same kind of operation at each node of such a data structure, often in some pre-defined order.

Iteratees are purely functional language constructs, which accept or reject data during the iterations.

Relation with recursion

Recursions and iterations have different algorithmic definitions, even though they can generate identical effects/results. The primary difference is that recursion can be employed as a solution without prior knowledge as to how many times the action will have to repeat, while a successful iteration requires that foreknowledge.

Some types of programming languages, known as functional programming languages, are designed such that they do not set up a block of statements for explicit repetition, as with the *for* loop. Instead, those programming languages exclusively use recursion. Rather than call out a block of code to be repeated a pre-defined number of times, the executing code block instead "divides" the work to be done into a number of separate pieces, after which the code block executes itself on each individual piece. Each piece of work will be divided repeatedly until the "amount" of work is as small as it can possibly be, at which point the algorithm will do that work very quickly. The algorithm then "reverses" and reassembles the pieces into a complete whole.

The classic example of recursion is in list-sorting algorithms, such as merge sort. The merge sort recursive algorithm will first repeatedly divide the list into consecutive pairs; each pair is then ordered, then each consecutive pair of pairs, and so forth until the elements of the list are in the desired order.

The code below is an example of a recursive algorithm in the Scheme programming language that will output the same result as the pseudocode under the previous heading.

```
(let iterate ((i 1) (a 0))
  (if (<= i 3)
    (iterate (+ i 1) (+ a i))
    (display a)))
```

Education

In some schools of pedagogy, iterations are used to describe the process of teaching or guiding students to repeat experiments, assessments, or projects, until more accurate results are found, or the student has mastered the technical skill. This idea is found in the old adage, "Practice makes perfect." In particular, "iterative" is defined as the "process of learning and development that involves cyclical inquiry, enabling multiple opportunities for people to revisit ideas and critically reflect on their implication."[1]

Unlike computing and math, educational iterations are not predetermined; instead, the task is repeated until success according to some external criteria (often a test) is achieved.

See also

- Recursion
- Fractal
- Brute-force search
- Iterated function
- Infinite compositions of analytic functions

References

1. Helen Timperley; Aaron Wilson; Heather Barrar; Irene Fung. "Teacher Professional Learning and Development: Best Evidence Synthesis Iteration [BES]" (http://www.oecd.org/edu/school/48727127.pdf) (PDF). OECD. p. 238. Retrieved 4 April 2013.