Computing Unit 4: Programming

Kurt Hornik

WIRTSCHAFTS UNIVERSITÄT WIEN VIENNA UNIVERSITY OF ECONOMICS AND BUSINESS

Outline

- Programming Elements
- Programming Tasks

Blocks

Using curly braces:

{ *EXPRS* }

EXPRS can be a list of expressions, separated either by newlines or semicolons.

Value of block is that of the last expression evaluated.

Conditionals


```
if( COND ) EXPR

if( COND ) EXPR1 else EXPR2
```

There is also `switch` to avoid deep nesting.

Loops

```
for( VAR in SEQ ) EXPR
```

This really performs *iteration* over the elements of *SEQ*, with *VAR* bound to these elements when evaluating *EXPR*.

```
while( COND ) EXPR
repeat EXPR
```

To advance to the next iteration: `next`

To terminate the loop: `break`

Functions

```
function ( ARGLIST ) EXPR
```

ARGLIST gives the *formals* (or arguments) of the function, as a comma-separated list of names, name = value pairs, or `...`.

EXPR gives the *body* of the function.

Outline

- Programming Elements
- Programming Tasks

Fibonacci Numbers

Defined by the second order recursion

$$f_1 = f_2 = 1, \qquad f_n = f_{n-1} + f_{n-2}, \quad n > 2.$$

See `https://en.wikipedia.org/wiki/Fibonacci_number`, which follows the more modern convention to start with index 0.

Fibonacci Numbers

Defined by the second order recursion

$$f_1 = f_2 = 1, \qquad f_n = f_{n-1} + f_{n-2}, \quad n > 2.$$

See `https://en.wikipedia.org/wiki/Fibonacci_number`, which follows the more modern convention to start with index 0.

So

$$\begin{aligned} f_3 &= f_2 + f_1 &= 2 \\ f_4 &= f_3 + f_2 &= 3 \\ f_5 &= f_4 + f_3 &= 5 \end{aligned}$$

etc. Alternatively, next element is the sum of its two predecessors:

$$({\color{red}1}, {\color{red}1}, 1+1 = {\color{red}2}, 1+2 = {\color{red}3}, 2+3 = {\color{red}5}, 3+5 = {\color{red}8}, 5+8 = {\color{red}13}, \ldots)$$

Sieve of Eratosthenes

See `https://en.wikipedia.org/wiki/Sieve_of_Eratosthenes`.

Start by listing all integers from 2 to *n* (here, 30):

```
     2  3  4  5  6  7  8  9 10 11 12 13 14 15
 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30
```

Sieve of Eratosthenes

See `https://en.wikipedia.org/wiki/Sieve_of_Eratosthenes`.

Start by listing all integers from 2 to *n* (here, 30):

```
    2  3  4  5  6  7  8  9 10 11 12 13 14 15
16 17 18 19 20 21 22 23 24 25 26 27 28 29 30
```

2 must be a prime number; all its multiples are not.

```
    2  3  4  5  6  7  8  9 10 11 12 13 14 15
16 17 18 19 20 21 22 23 24 25 26 27 28 29 30
```

Sieve of Eratosthenes

3 must be a prime number; all its multiples are not.

```
    2  3  4  5  6  7  8  9 10 11 12 13 14 15
16 17 18 19 20 21 22 23 24 25 26 27 28 29 30
```

Sieve of Eratosthenes

3 must be a prime number; all its multiples are not.

	2	3	4	5	6	7	8	9	10	11	12	13	14	15
16	17	18	19	20	21	22	23	24	25	26	27	28	29	30

5 must be a prime number; all its multiples are not.

	2	3	4	5	6	7	8	9	10	11	12	13	14	15
16	17	18	19	20	21	22	23	24	25	26	27	28	29	30

Sieve of Eratosthenes

3 must be a prime number; all its multiples are not.

	2	3	4	5	6	7	8	9	10	11	12	13	14	15
16	17	18	19	20	21	22	23	24	25	26	27	28	29	30

5 must be a prime number; all its multiples are not.

	2	3	4	5	6	7	8	9	10	11	12	13	14	15
16	17	18	19	20	21	22	23	24	25	26	27	28	29	30

7 must be a prime number etc etc.

	2	3	4	5	6	7	8	9	10	11	12	13	14	15
16	17	18	19	20	21	22	23	24	25	26	27	28	29	30

Newton's method

How can we find a root of a differentiable function f?

Suppose we have a "candidate value" x_n.

Close to x_n, we can approximate $f(x)$ by its tangent at x_n:

$$f(x) \approx g_n(x) = f(x_n) + f'(x_n)(x - x_n).$$

Idea of Newton's method: try approximating roots of f by the root of g_n!

Newton's method

g_n has a single root unless $f'(x_n) = 0$:

$$\begin{aligned} g_n(x) = 0 \quad & \Leftrightarrow \quad f'(x_n)(x - x_n) = -f(x_n) \\ & \Leftrightarrow \quad x - x_n = -\frac{f(x_n)}{f'(x_n)} \\ & \Leftrightarrow \quad x = x_n - \frac{f(x_n)}{f'(x_n)} \end{aligned}$$

Repetitively applying this idea gives the recursion

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

with suitable starting point x_0.