

Dr. Straight's Maple Examples

Example I: Numeric Operations and Functions

Functions Covered: `ceil`, `denom`, `evalf`, `floor`, `frac`, `Fraction`, `ifactor`, `ifactors`, `igcd`, `igcdex`, `ilcm`, `iquo`, `irem`, `mod`, `numer`, `round`, `sign`, `trunc`

The first thing to keep in mind about Maple is that it prefers to work with numbers *exactly*, rather than with approximations. Hence, if we enter

```
> t := sqrt(5);
```

$$t := \sqrt{5} \quad (1)$$

Maple stores the exact value of the square root of 5 in `t`. If we want a decimal approximation to the square root of 5, we use the **evalf** command, which means, "evaluate as floating-point."

```
> evalf(t);
```

$$2.236067977 \quad (2)$$

By default, `evalf` gives 10 digits; however, there is an optional second argument that specifies the number of digits, for example

```
> evalf(t, 12);
```

$$2.23606797750 \quad (3)$$

Here are several more examples illustrating the use of `evalf`:

```
> m := (-3)^4;
```

$$m := 81 \quad (4)$$

```
> k := 27^(1/3);
```

$$k := 27^{1/3} \quad (5)$$

```
> k := evalf(27^(1/3));
```

$$k := 3.000000000 \quad (6)$$

```
> k := simplify(27^(1/3));
```

$$k := 3 \quad (7)$$

```
> f := 37/104;
```

$$f := \frac{37}{104} \quad (8)$$

```
> evalf(f);
```

$$0.3557692308 \quad (9)$$

```
> evalf(f, 20);
```

$$0.35576923076923076923 \quad (10)$$

Fractions can be constructed using the division operator, as seen above, or by using the **Fraction** constructor. The functions `numer` and `denom` return the numerator and denominator of a given fraction, respectively.

```
> f := 111/-312;
```

$$f := -\frac{37}{104} \quad (11)$$

```
> f := Fraction(111,-312);
```

$$\quad (12)$$

$$f := -\frac{37}{104} \quad (12)$$

> $n := \text{numer}(f);$

$$n := -37 \quad (13)$$

> $d := \text{denom}(f);$

$$d := 104 \quad (14)$$

Note that a fraction is always stored in lowest terms, with a positive denominator.

The **iquo** (for "integer quotient") and **irem** (for "integer remainder") functions are used to perform integer division. For example:

> $\text{iquo}(31, 7);$

$$4 \quad (15)$$

> $\text{irem}(31, 7);$

$$3 \quad (16)$$

> $\text{iquo}(-31, 7);$

$$-4 \quad (17)$$

> $\text{irem}(-31, 7);$

$$-3 \quad (18)$$

> $\text{iquo}(31, -7);$

$$-4 \quad (19)$$

> $\text{irem}(31, -7);$

$$3 \quad (20)$$

> $\text{iquo}(-31, -7);$

$$4 \quad (21)$$

> $\text{irem}(-31, -7);$

$$-3 \quad (22)$$

If, like me, you were taught that a remainder is never negative, then you may take issue with Maple's results that, when -31 is divided by 7, the quotient is -4 and the remainder is -3. For this reason, I recommend avoiding use of the **iquo** and **irem** functions when the dividend is negative. For computing remainders, I recommend using the **mod** operator:

> $31 \text{ mod } 7;$

$$3 \quad (23)$$

> $-31 \text{ mod } 7;$

$$4 \quad (24)$$

> $31 \text{ mod } -7;$

$$3 \quad (25)$$

> $-31 \text{ mod } -7;$

$$4 \quad (26)$$

Here are several examples related to "rounding" numbers:

> $\text{round}(\text{sqrt}(7));$

$$3 \quad (27)$$

> $\text{round}(-\text{sqrt}(7));$

$$-3 \quad (28)$$

> $\text{round}(2.5);$

$$3 \quad (29)$$


```
> igcd(12, 54, 72);
6 (47)
```

```
> lcm(54, 72);
216 (48)
```

```
> lcm(10, 54, 72);
1080 (49)
```

OK, so `igcd` simply implements the Euclidean algorithm. What about the extended Euclidean algorithm? Not to worry, Maple has it covered -- the **`igcdex`** function has two additional arguments, s and t , say, so that

`igcdex(a,b,s,t)`

not only returns $d = \gcd(a,b)$, but also stores in s and t values such that $d = as + bt$

Here's an example:

```
> igcdex(2544, 5436, s, t);
12 (50)
```

```
> s;
203 (51)
```

```
> t;
-95 (52)
```

Finally, we have the **`ifactor`** function for finding prime factorizations.

```
> ifactor(360);
(2)3 (3)2 (5) (53)
```

```
> ifactor(-360);
-(2)3 (3)2 (5) (54)
```

Even more useful is the **`ifactors`** function:

```
> ifactors(360);
[1, [[2, 3], [3, 2], [5, 1]]] (55)
```

```
> ifactors(-360);
[-1, [[2, 3], [3, 2], [5, 1]]] (56)
```

Note that, for an integer m (not 0), `ifactors(m)` returns a list. The first element of this list is $\text{sign}(m)$. Assuming $|m| > 1$, the second element of the list is another list, made up of two-element sublists. Each of these sublists has the form $[p,k]$, where p is a prime factor of m and k is the exponent on p that appears in the prime factorization of m . Also, these sublists are ordered according to the prime factors.

Given a list c , the notation $c[i]$ denotes the i th element of c . For example:

```
> c := ifactors(360)[2];
c := [[2, 3], [3, 2], [5, 1]] (57)
```

```
> c[2];
[3, 2] (58)
```

```
> c[2][1];
3 (59)
```

Challenge Exercise: Assuming the variable n holds a positive integer that is not a power of 2 (that is, n contains at least one odd prime factor), write a Maple command that returns the smallest odd prime factor of n .

Caution: Sometimes, you may get a strange error when trying to use a particular variable, say v . If this happens, try checking whether v is already defined by using the command `?v`. If v has a value, and you

need it to be "free," use the command " $v := v$ " to clear v . Alternately (and with caution), you may issue the "restart" command, which clears Maple's memory.

>