```
>
>
```
about keyword parameters
```
> top:=proc(x,{thing:=2})
    print("you gave me x=",x,"and thing is",thing);
  end:
```
```
> top(3);
```
"you gave me x=", 3, "and thing is", 2 **(1)**
```
> top(4, thing=rabbit);
```
"you gave me x=", 4, "and thing is", *rabbit* **(2)**
```
> mid:=proc(y,{thing:=2})
    print("you gave mid y=",y,"and thing is",thing);
  end:
> othertop:=proc(x,{thing:=2})
    print("you gave me x=",x,"and thing is",thing);
    print("what does mid get?");
     mid(x);
  end:
```
```
> othertop(3);
```
"you gave me x=", 3, "and thing is", 2
"what does mid get?"
"you gave mid y=", 3, "and thing is", 2 **(3)**
```
> othertop(3, thing=5);
```
"you gave me x=", 3, "and thing is", 5
"what does mid get?"
"you gave mid y=", 3, "and thing is", 2 **(4)**
```
> othertop := proc(x, {thing := 2})
    print("you gave me x=", x, "and thing is", thing);
    print("what does mid get?");
    mid(x, thing=5); # maple sees mid(x,72=5)
  end:
```
```
> othertop(3, thing=7);
```
"you gave me x=", 3, "and thing is", 7
"what does mid get?"
"you gave mid y=", 3, "and thing is", 2 **(5)**
```
> debug(othertop); debug(mid);
```
othertop
mid **(6)**
```
> othertop(3, thing=7);
{--> enter othertop, args = 3, thing = 7
```
"you gave me x=", 3, "and thing is", 7
"what does mid get?"
```
{--> enter mid, args = 3, 7 = 5
```
"you gave mid y=", 3, "and thing is", 2
```
<-- exit mid (now in othertop) = }
<-- exit othertop (now at top level) = }
```

```
> othertop:=proc(x,{thing:=2})
      print("you gave me x=",x,"and thing is",thing);
      print("what does mid get?");
      unassign('thing');
      mid(x,thing=5);
  end:
```

```
> debug(othertop); debug(mid); othertop(3, thing = 7);
                                    othertop
                                      mid
{--> enter othertop, args = 3, thing = 7
                    "you gave me x=", 3, "and thing is", 7
                              "what does mid get?"
<-- ERROR in othertop (now at top level) = cannot unassign `%1'
(argument must be assignable), 7}
Error, (in unassign) cannot unassign `7' (argument must be
assignable)
```

```
> tip:=proc(x,{thing:=2})
      print("you gave me x=",x,"and thing is",thing);
      print("options has",_options);
      print("_options['thing'] is ",_options['thing']);
      print("what does mid get?");
      mid(x);
  end:
```

```
> tip(3, thing = hello, thang = hoho);
                    "you gave me x=", 3, "and thing is", hello
                          "options has", thing = hello
                    "_options['thing'] is ", thing = hello
                              "what does mid get?"
{--> enter mid, args = 3
                     "you gave mid y=", 3, "and thing is", 2
<-- exit mid (now in tip) = }
```

```
> tip := proc(x, {thing := 2})
    print("you gave me x=", x, "and thing is", thing);
    print("options has", _options);
    print("_options['thing'] is ", _options['thing']);
    print("what does mid get?");
    mid(x, _options['thing']);
  end:
```

```
> tip(3, thing = "yay");
                    "you gave me x=", 3, "and thing is", "yay"
                          "options has", thing = "yay"
                    "_options['thing'] is ", thing = "yay"
                              "what does mid get?"
{--> enter mid, args = 3, thing = yay
                    "you gave mid y=", 3, "and thing is", "yay"
<-- exit mid (now in tip) = }
```

```
> tap:=proc(x,{thing:=2})
      print("you gave me x=",x,"and thing is",thing);
```

```
      print("options has",_options);
      print("_options['thing'] is ",_options['thing']);
      print("what does mid get?");
      mid(x, 'thing'=7);
   end:
> tap(3);
                          "you gave me x=", 3, "and thing is", 2
                                "options has", thing = 2
                            "_options['thing'] is ", thing = 2
                                "what does mid get?"
{--> enter mid, args = 3, 2 = 7
                          "you gave mid y=", 3, "and thing is", 2
<-- exit mid (now in tap) = }
> tap:=proc(x,{thing:=2})
      print("you gave me x=",x,"and thing is",thing);
      print("options has",_options);
      print("_options['thing'] is ",_options['thing']);
      print("what does mid get?");
      mid(x, ':-thing'=7);
   end:
> tap(3);
                          "you gave me x=", 3, "and thing is", 2
                                "options has", thing = 2
                            "_options['thing'] is ", thing = 2
                                "what does mid get?"
{--> enter mid, args = 3, thing = 7
                          "you gave mid y=", 3, "and thing is", 7
<-- exit mid (now in tap) = }
>
>
> fact := proc(n :: posint)
    if (n = 1 ) then return(1); fi;
    return(n·fact(n − 1));
   end:
> fact(5);
                                      120                                (7)
> debug(fact);
                                      fact                               (8)
> fact(5) ;
{--> enter fact, args = 5
{--> enter fact, args = 4
{--> enter fact, args = 3
{--> enter fact, args = 2
{--> enter fact, args = 1
<-- exit fact (now in fact) = 1}
<-- exit fact (now in fact) = 2}
<-- exit fact (now in fact) = 6}
<-- exit fact (now in fact) = 24}
<-- exit fact (now at top level) = 120}
                                      120                                (9)
```

```
>
>
>
>
>
>
> 3^5 mod 7;
                                    5                                    (10)
> 3^5
                                   243                                   (11)
> 3^400 mod 7
                                    4                                    (12)
> 3^400
705507910865533257124642715759347962165079496127873157628712232092620855515821\
    9341565792985294471341581549523348253559118669297930718245666941450844545 3\
    5257027960285323760313192443283334088001                             (13)
> 400 mod 6;
                                    4                                    (14)
> nextprime(1234567);
                                 1234577                                 (15)
> 5^1234576 mod 1234577;
                                    1                                    (16)
> nextprime(
      1111111111111111111111111111111111111111111111111111111111111111111111111111111111111111\
      1111111111111111111111111111111111111111111111111111111111111111111111111111111111111111\
      1111111111111111111111111111111111111111111111111111111111111111111111111111111111111111\
      11111);
1111111111111111111111111111111111111111111111111111111111111111111111111111111111111111\
    1111111111111111111111111111111111111111111111111111111111111111111111111111111111111111\
    1111111111111111111111111111111111111111111111111111111111111111111111111111111111111111\
    12551                                                                (17)
> ifactor(
      1111111111111111111111111111111111111111111111111111111111111111111111111111111111111111\
      1111111111111111111111111111111111111111111111111111111111111111111111111111111111111111\
      1111111111111111111111111111111111111111111111111111111111111111111111111111111111111111\
      1111);
Warning, computation interrupted
> 5^9 mod 10;
                                    5                                    (18)
> 5&^1111111111111111111111111111111111111111111111111111111111111111111111111
  1111111111111111111111111111111111111111111111111111111111111111111111111
  1111111111111111111111111111111111111111111111111111111111111111111111111
  11111111111111111111111111111110 mod
  1111111111111111111111111111111111111111111111111111111111111111111111111
  1111111111111111111111111111111111111111111111111111111111111111111111111
  1111111111111111111111111111111111111111111111111111111111111111111111111
```

    1111111111111111111111111111111;

7004393749375664239659849039535188975604219418059769241523888748622551681934 6\
    1780049085549438848389627923696613775134559368422743374500699891601679099 7\
    9006423363864026021510855709150734405021832664615037714112591248052640501 0\
    554 **(19)**

> $5^{1111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111\backslash 1111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111\backslash 1111111111111111111111110}$ **mod** 1111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111\
1111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111\
11111111111111111111111111111

Error, numeric exception: overflow

>