EXAMPLE FILE FOR MERGETEX

PAUL ZINN-JUSTIN

1. Introduction

some basic examples:

```
i1 : R=QQ[x,y]; factor(x^3-y^3)
```

o2 = $(x-y)\,(x^2+x\,y+y^2)$

```
o2 : Expression of class Product
i3 : res coker vars R
```

o3 = $R^1 \xleftarrow{\begin{pmatrix} x & y \end{pmatrix}}_{0} R^2 \xleftarrow{\begin{pmatrix} -y \\ x \end{pmatrix}}_{1} R^1 \xleftarrow[2]{0} 0_{3}$

```
o3 : ChainComplex
i4 : OO_(Proj(R/(x^3-y^3)))^{1,2}
```

o4 = $\mathcal{O}^1_{\mathtt{Proj}\left(\frac{R}{x^3-y^3}\right)}(1) \oplus \mathcal{O}^1_{\mathtt{Proj}\left(\frac{R}{x^3-y^3}\right)}(2)$

```
o4 : coherent sheaf on Proj (R/(x^3-y^3)), free
i5 : matrix {{1,2},{3,4}}
```

o5 = $\begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}$

```
o5 : Matrix ZZ^2 <-- ZZ^2
```

The code can also be inline: `gcd(1300,75)`. More:

```
i6 : 318/46
```

o6 = $\frac{159}{23}$

```
o6 : QQ
i7 : exp 3.73767
```

o7 = 42.0000160321016

```
o7 : RR (of precision 53)
```

strings and nets:

```
i8 : "hehe"
o8 = hehe
i9 : ( "haha123456789"
     ||"hoho!@#$%^&*(")
o9 =  haha123456789
      hoho!@#$%^&*(
```

```
i10 : {oo,ooo}
```

$$\texttt{o10} = \left\{ \begin{array}{l} \texttt{haha123456789} \\ \texttt{hoho!@\#\$\%\^{}\&*(} \end{array} , \texttt{hehe} \right\}$$

```
o10 : List
```

printing:

```
i11 : for i from 1 to 8 do print((i+ii)^2)
```

$2\,\mathbf{i}$
$3 + 4\,\mathbf{i}$
$8 + 6\,\mathbf{i}$
$15 + 8\,\mathbf{i}$
$24 + 10\,\mathbf{i}$
$35 + 12\,\mathbf{i}$
$48 + 14\,\mathbf{i}$
$63 + 16\,\mathbf{i}$

2. Reusing output

The output `o5` is $\begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}$. The nonexistent output `o11` is .

3. Inputting from external file

Some more code:

```
i12 : -- a test file
      R=QQ[x,y,z]
```

$$\texttt{o12} = R$$

```
o12 : PolynomialRing
i13 : poincare ideal(x^2+y^2,x^3+z^3)
```

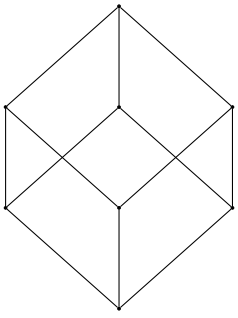
$$\texttt{o13} = 1 - T^2 - T^3 + T^5$$

$$\texttt{o13} : \mathbb{Z}[T]$$

4. Packages

packages that have a `tex` output will work:

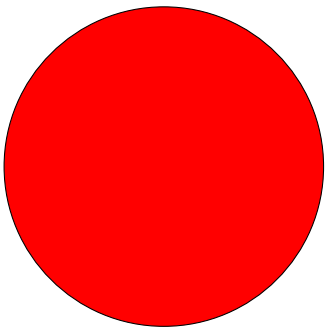
```
i14 : needsPackage "Posets";
i15 : booleanLattice 3
```



```
o15 =
```

```
o15 : Poset
i16 : needsPackage "VectorGraphics";
i17 : Circle{"fill"=>"red"}
```

o17 =



```
o17 : Circle
```

5. Changing key/values

```
i18 : "some␣weird␣spacing␣␣and␣string␣style"
o18 = some weird spacing and string style
```

6. Preventing output

This code shouldn't be processed:

```
f = i -> (
            print i;
            i+1
          )
```

7. Help

```
i19 : help cohomology
o19 =
```

cohomology – general cohomology functor

Synopsis

- Optional inputs:
  - Degree => ..., default value 0,

Description

`cohomology` – a method name available for computing expressions of the forms `HH^i(X)` and `HH^i(M,N)`.

If it is intended that `i` be of class `ZZ`, `M` be of class `A`, and `N` be of class `B`, then the method can be installed with

```
cohomology(ZZ, A, B) := opts -> (i,M,N) -> ...
```

See also

- `homology` – general homology functor
- `HH` – general homology and cohomology functor
- `ScriptedFunctor` – the class of all scripted functors

Ways to use `cohomology` :

- `HH^ZZ ChainComplex` – cohomology of a chain complex
- `HH^ZZ ChainComplexMap` – cohomology of a chain complex map
- `HH^ZZ Module` – local cohomology of a module
- `HH^ZZ SheafOfRings` – cohomology of a sheaf of rings on a projective variety
- `HH^ZZ SimplicialMap` – Compute the induced map on cohomology of a simplicial map.
- `HH^ZZ SumOfTwists` – coherent sheaf cohomology module
- `"HH^ZZ CoherentSheaf"` – see `HH^ZZ(ProjectiveVariety,CoherentSheaf)` – cohomology of a coherent sheaf on a projective variety
- `HH^ZZ(ProjectiveVariety,CoherentSheaf)` – cohomology of a coherent sheaf on a projective variety
- `"HH^ZZ SimplicialComplex"` – see `HH^ZZ(SimplicialComplex,Ring)` – compute the reduced cohomology of an abstract simplicial complex
- `HH^ZZ(SimplicialComplex,Ring)` – compute the reduced cohomology of an abstract simplicial complex
- `HH^ZZ(SimplicialComplex,SimplicialComplex)` – compute the relative homology of two simplicial complexes

For the programmer

The object `cohomology` is a `method function with options`.

8. Tricky examples

. . . for testing purposes only

```
i20 : -- some tricky examples
```

A bunch of complicated cases: a multi-line example

```
      f = i -> (
      -- that's dumb
      i+1
      )
o20 = f
o20 : FunctionClosure
```

and another weirder one:

```
i21 : I=ideal 0; f = i -> (
o21 : Ideal of ℤ
      i+1)
o22 = f
o22 : FunctionClosure
```

finally:

```
i23 : a=1;b=2;
i25 : c=3;
```

That last one has no output.