Example of GAP SCSCP client connecting to Python 3 SCSCP server

In this example GAP SCSCP client communicates with the Python 3 SCSCP server. The Python code is based on https://github.com/OpenMath/py-scscp/blob/master/demo_server.py (https://github.com/OpenMath/py-scscp/blob/master/demo_server.py)

Simple calls

In [1]:

```
EvaluateBySCSCP("plus",[2,2],"localhost",26133:cd:="arith1").object
```

```
4
```

In [2]:

```
EvaluateBySCSCP("plus",[[1,2],[3,4]],"localhost",26133:cd:="arith1").object
```

```
[ 1, 2, 3, 4 ]
```

In Python, addition of lists and strings is their concatenation

In [3]:

```
EvaluateBySCSCP("plus",["abc","def"],"localhost",26133:cd:="arith1").object
```

```
"abcdef"
```

Using NumPy linear algebra tools

In the next example, we extend Python server to offer some procedures from the NumPy package for scientific computing (http://www.numpy.org/ (http://www.numpy.org/)). To do that, we need only to add several more lines to the Python script to run the server:

```
import numpy

CD_SCSCP_TRANSIENT1 = {
    'numpy.linalg.det'          : numpy.linalg.det,
    'numpy.linalg.matrix_rank' : lambda x: int(numpy.linalg.matrix_rank
(x)),
}
```

- Compute determinant and rank of a random 5x5 matrix

In [4]:

```
m:=RandomMat(5,5);
```

```
[ [ 1, 0, -1, -1, -1 ], [ 1, -1, 1, -2, -1 ], [ -2, 0, -1, 2, -2 ],
  [ -1, 2, -3, -1, 3 ], [ 0, -2, 1, -4, 0 ] ]
```

In [5]:

```
EvaluateBySCSCP("numpy.linalg.det",[m],"localhost",26133:OMignoreMatrices).objec
t;
```

```
-36.
```

In [6]:

```
EvaluateBySCSCP("numpy.linalg.matrix_rank",[m],"localhost",26133:OMignoreMatrice
s).object;
```

```
5
```

Let's try with matrices of larger dimensions

In [7]:

```
EvaluateBySCSCP("numpy.linalg.det",[RandomMat(50,50)],"localhost",26133:OMignore
Matrices).object;
```

```
-7.67794e+49
```

In [8]:

```
EvaluateBySCSCP("numpy.linalg.matrix_rank",
[RandomMat(50,50)],"localhost",26133:OMignoreMatrices).object;
```

```
50
```

Using NumPy to calculate complex roots of polynomials

Similarly, on the Python server we export another function that calculates (complext) roots of univariate polynomials and returns a list of their real and imaginary parts:

```
def polyroots( coeffs ):
    f = numpy.polynomial.polynomial.Polynomial( coeffs )
    r = f.roots()
    return [ [x.real,x.imag] for x in r]
```

- create polynomials with integer roots

In [9]:

```
x:=X(Rationals,"x"); f:=(x-10)*(x-1)*(x+5);
```

```
<object>
```

- calculate roots with GAP

In [10]:

```
RootsOfUPol(f);
```

```
Error, Variable: 'f' must have a value
```

- check that Python results agree

In [11]:

```
coeffs:=CoefficientsOfUnivariatePolynomial(f)
```

```
Error, Variable: 'f' must have a value
```

In [12]:

```
EvaluateBySCSCP("polyroots",[ coeffs ],"localhost",26133:OMignoreMatrices).objec
t;
```

```
Error, Variable: 'coeffs' must have a value
```

- But GAP can not compute (approximations of) complex roots of another polynomial

In [13]:

```
RootsOfUPol(1+2*x+3*x^2);
```

```
[  ]
```

- However, Python with the help of NumPy is capable of doing this

In [14]:

```
coeffs := CoefficientsOfUnivariatePolynomial(1+2*x+3*x^2)
```

```
[ 1, 2, 3 ]
```

In [15]:

```
EvaluateBySCSCP("polyroots",[ coeffs ],"localhost",26133:OMignoreMatrices).objec
t;
```

```
[ [ -0.333333, -0.471405 ], [ -0.333333, 0.471405 ] ]
```