Example of SCSCP client in Python2 connecting to GAP server

In [1]:
```
from scscp import SCSCPCLI
```

- Establishing connection with the demo SCSCP server

In [2]:
```
c = SCSCPCLI('scscp.gap-system.org')
```

- Ask for the list of supported procedures

In [3]:
```
c.heads
```

Out[3]:
```
{'scscp_transient_1': ['SCSCPStartTracing', 'Addition', 'IO_UnpickleSt
ringAndPickleItBack', 'NrConjugacyClasses', 'ConwayPolynomial', 'Small
Group', 'GroupIdentification', 'AutomorphismGroup', 'IdGroup512ByCod
e', 'Phi', 'Factorial', 'GnuExplained', 'MathieuGroup', 'TransitiveGro
up', 'PrimitiveGroup', 'Multiplication', 'NextUnknownGnu', 'Identity',
'IsPrimeInt', 'Gnu', 'Determinant', 'LatticeSubgroups', 'Length', 'Mat
rixMultiplication', 'SCSCPStopTracing', 'AlternatingGroup', 'Symmetric
Group', 'IdGroup', 'SylowSubgroup', 'GnuWishlist', 'Size']}
```

- A simplest "ping-pong" test which sends an object to the server and gets it back

In [4]:
```
c.heads.scscp_transient_1.Identity([1])
```

Out[4]:
```
1
```

- Examples of some procedure calls

In [5]:
```
c.heads.scscp_transient_1.Factorial([10])
```

Out[5]:
```
3628800
```

In [6]:
```
c.heads.scscp_transient_1.IsPrimeInt([2**16+1])
```

Out[6]:
```
True
```

- In the next example, we calculate the symmetric group of degree 3

In [7]:
```
g = c.heads.scscp_transient_1.SymmetricGroup([3])
```

- This group does not map to an object defined in Python, so it is stored in its internal representation

```
In [8]: g
```

```
Out[8]: OMApplication(OMSymbol('group', 'permgp1', id=None, cdbase=None), [OMS
        ymbol('right_compose', 'permutation1', id=None, cdbase=None), OMApplic
        ation(OMSymbol('permutation', 'permut1', id=None, cdbase=None), [OMInt
        eger(2, id=None), OMInteger(3, id=None), OMInteger(1, id=None)], id=No
        ne, cdbase=None), OMApplication(OMSymbol('permutation', 'permut1', id=
        None, cdbase=None), [OMInteger(2, id=None), OMInteger(1, id=None)], id
        =None, cdbase=None)], id=None, cdbase=None)
```

- But we can use it as an argument in SCSCP procedure calls, for example, to find its order and the catalogue number in the GAP Small Groups Library

```
In [9]: c.heads.scscp_transient_1.Size([g])
```

```
Out[9]: 6
```

```
In [10]: c.heads.scscp_transient_1.NrConjugacyClasses([g])
```

```
Out[10]: 3
```

```
In [11]: c.heads.scscp_transient_1.IdGroup([g])
```

```
Out[11]: [6, 1]
```

However, this is not very efficient:

- OpenMath encoding for an object may be quite verbose
- Sending it to the GAP server will create a new object instead of using the existing one
- There may be situations when GAP may not be able to convert the result into OpenMath

For such scenarious, SCSCP specification defines remote objects

- Create a remote copy of the alternating group of degree 5 and ask to return a reference

```
In [12]: g=c.heads.scscp_transient_1.AlternatingGroup([5],cookie=True)
```

```
In [13]: g
```

```
Out[13]: OMReference('scscp://chrystal.mcs.st-andrews.ac.uk:26133/TEMPVarSCSCPJ
         XmNfISQ', id=None)
```

- The reference could be used as an arguments of the procedure call
- While the client does not "know" what this groups is, it can certainly "understand" various numerical properties of this group

```
In [14]: c.heads.scscp_transient_1.Size([g])
```

```
Out[14]: 60
```

- Now we will create an automorphism group of the given group and receive another reference

```
In [15]: a = c.heads.scscp_transient_1.AutomorphismGroup([g],cookie=True)
```

```
In [16]: a
```

```
Out[16]: OMReference('scscp://chrystal.mcs.st-andrews.ac.uk:26133/TEMPVarSCSCPT
         XMaSBJM', id=None)
```

- And then we are able to investigate various numerical properties of that group

```
In [17]: c.heads.scscp_transient_1.Size([a])
```

```
Out[17]: 120
```

```
In [18]: c.heads.scscp_transient_1.NrConjugacyClasses([a])
```

```
Out[18]: 7
```

```
In [19]: c.heads.scscp_transient_1.IdGroup([a])
```

```
Out[19]: [120, 34]
```

- Close the connection

```
In [20]: c.quit()
```