Common Syntax Structures

```
exp [any expression], stmt [(sequence of) command(s)]
```

Note: Indentation is important for control sequences!

### Assignment
```
a = 1
b, c = 1, 2
c = [1,2,3]; c[1] = 4
```

### Output
```
print(expr...) print("test")
```

### Comment
```
# single line
"""multi
line"
```

### Selection
```
If boolean_exp:
| stmt
| [elif boolean_exp: stmt] else:
| [else: stmt]
```

### Repetition
```
while(boolean_exp):
| stmt
```

### Traversal
```
for var in obj:
| stmt
```

### Loop
```
for i in range(0,5):
| print(i)
```

### Exception Handling
```
try:
| stmt ...
| except [exc_type] [var]:
| stmt ...
```

### Function Definition
```
def fnname(params):
| """doc-string"
```

### Call
```
ret = fnname(args)
```

### Common Built-in Functions
```
abs(x)  Absolute value of x
float(x), int(x)  Convert x to float / int (if possible)
len(s)  Number of items in sequence (list, tuple, ...)
str(obj)  String representation of obj
range(start, stop, [step])  A list [x, x+1, x+2, ..., y-1] (y excluded)
dict()  Empty dictionary
list()  Empty list
tuple()  Empty tuple
```

### Modulo Import Example
```
Help
help(m)  Display help for module, function...
pluginHelp("name")  Display information for plugin
filterHelp("name")  Display information for itom-filter
widgetHelp("name")  Display information for widget in plugin
dir(m)  Display names in module m
```

### Common Data Types
```
tuple()  Empty tuple
list()  Empty list
dict()  Empty dictionary
str(obj)  String representation of obj
len(s)  Number of items in sequence (list, tuple, ...)
float(x), int(x)  Convert x to float / int (if possible)
```

### Common Built-in Functions
```
abs(x)  Absolute value of x
```

### Common Functions of Module math (from math import *)
```
cos(x), sin(x), tan(x)  Cosine, sine, tangent of x radians
cosh(x), sinh(x), tanh(x)  Hyperbolic cosine, sine, tangent of x
sqrt(x)  Positive square root of x
degrees(x), radians(x)  Convert from rad to deg, deg to rad
exp(x)  e ** x
floor(x)  Largest whole number <= x
pow(x,y)  x ** y
pi  Math constant π (15 sig figs)
e  Math constant e (15 sig figs)
```

### Common List (L) and Tuple (T) Methods
```
LT[idx], LT[idx1:idx2]  get items or slice of items from list/tuple
LT.count(obj)  number of occurrences of obj in LT
LT.index(obj)  index of first occurrence of obj in LT; raises ValueError if does not occur
L[idx] = obj  assigns new value to index (list only)
L.append(obj)  Appends obj to end of list L
L.remove(obj)  Removes first occurrence of obj from L
```

### Common List (L) or Tuple (T) Methods, (LT both)
```
LT[idx], LT[idx1:idx2]  get items or slice of items from list/tuple
LT.count(obj)  number of occurrences of obj in LT
LT.index(obj)  index of first occurrence of obj in LT; raises ValueError if does not occur
L[idx] = obj  assigns new value to index (list only)
L.append(obj)  Appends obj to end of list L
L.remove(obj)  Removes first occurrence of obj from L
```

### Common Dictionary (D) Methods
```
D["key"]  returns value corresponding to key
D["key"] = obj  replaces/adds obj under given key
“key” in D  True if key exists in D, else False
D.clear()  clears dictionary
D.keys()  Returns list of D’s keys
D.values()  Returns list of D’s values
```

### Formatting Numbers as Strings

**Syntax:** \%[width][.precision type] % expression

**width** (optional)  total width (+/-: right/left aligned)
**precision** (optional)  specified digits of float precision
**type (required)**  \d (int), \f (float), \s (string), e (exp. Notation)

**Examples:**
```
"%6d" % 123 -> ...123
"%04d" % 1 -> ...0001
"%8.2f" % 456.789 -> .456.79
"%8.2e" % 456.789 -> 4.57e+02
```

---

1 only available in itom
## Working with dataIO-Devices (Grabber, AD-Converter,...)

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pluginHelp(&quot;name&quot;)</td>
<td>Prints information about plugin</td>
</tr>
<tr>
<td>dataIO(&quot;name&quot;,params)</td>
<td>Creates obj (instance) of device</td>
</tr>
<tr>
<td>obj.getParam(&quot;name&quot;)</td>
<td>Returns value of parameter</td>
</tr>
<tr>
<td>obj.setPosition(idx1,pos1,...)</td>
<td>Sets parameter to pos</td>
</tr>
<tr>
<td>obj.setPositionRelative(idx1,pos1,...)</td>
<td>Relatively moves axis</td>
</tr>
<tr>
<td>obj.getName()</td>
<td>Returns name of device</td>
</tr>
<tr>
<td>obj.getImage()</td>
<td>Returns acquired image</td>
</tr>
<tr>
<td>obj.setAutoGrabbing(bool)</td>
<td>Enables/disables continuous grab for connected live views</td>
</tr>
<tr>
<td>obj.getImage()</td>
<td>Acquires image</td>
</tr>
<tr>
<td>obj.stopDevice()</td>
<td>Stops device (camera, sensor, etc.)</td>
</tr>
<tr>
<td>obj.startDevice()</td>
<td>Starts device (camera, sensor, etc.)</td>
</tr>
<tr>
<td>obj.setParam(&quot;name&quot;,val)</td>
<td>Sets parameter to val</td>
</tr>
<tr>
<td>obj.getParam(&quot;name&quot;)</td>
<td>Returns value of parameter</td>
</tr>
<tr>
<td>dataIO(&quot;name&quot;,params)</td>
<td>Creates obj (instance) of device</td>
</tr>
</tbody>
</table>

## Working with actuator-Devices (Motors, Stages,...)

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pluginHelp(&quot;name&quot;)</td>
<td>Prints information about plugin</td>
</tr>
<tr>
<td>actuator(&quot;name&quot;,params)</td>
<td>Creates instance of device</td>
</tr>
<tr>
<td>obj.getPosition()</td>
<td>Returns current position of all given axes</td>
</tr>
<tr>
<td>obj.getPosRel(idx1,idx2,...)</td>
<td>Relatively moves axis with given parameters</td>
</tr>
<tr>
<td>obj.getPosAbs(idx1,idx2,...)</td>
<td>Moves axis with given parameters</td>
</tr>
</tbody>
</table>

## Working with itom-Filters

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>filterHelp(&quot;name&quot;)</td>
<td>Lists available filters</td>
</tr>
<tr>
<td>ret=filter(&quot;name&quot;,param1,...)</td>
<td>Calls filter with the given parameters</td>
</tr>
</tbody>
</table>

## Plots

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>plot(dObj)</td>
<td>1D or 2D plot of dObj</td>
</tr>
<tr>
<td>liveImage(dataIO-instance)</td>
<td>Live view of camera</td>
</tr>
</tbody>
</table>

## Common DataObject and Numpy.Array Data Types

- **"uint8", "int8", "uint16", "int16", "uint32", "int32"**
  - **(Un-)Signed integer**: 8,16,32 bit
- **"float32", "float64"**
  - Floating point numbers
- **"complex64", "complex128"**
  - Complex values (64 = 2x32 bit)

## Numpy.array (import numpy as np, np.array), DataObject (import itom, itom.dataObject)

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>arr=np.random.array([2,3],type='uint8')</td>
<td>Create a 2x3 array with type uint8</td>
</tr>
<tr>
<td>arr=np.array([(1,2,3),[4,5,6]])</td>
<td>Create a 2x3 array [1,2,3; 4,5,6]</td>
</tr>
<tr>
<td>arr=dObj.dataObject([2,3],type='uint8')</td>
<td>Convert np.array to a DataObject</td>
</tr>
</tbody>
</table>

## Python/Numpy-Arrays/DataObjects

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>np.linspace(1,3,4)</td>
<td>Creates an array of 4 equally spaced samples between 1 and 3, inclusive</td>
</tr>
<tr>
<td>x[y] = np.meshgrid(0:2,1:5)</td>
<td>Two 2D arrays: one of x values, the other of y values</td>
</tr>
<tr>
<td>np.linalg.inv(a)</td>
<td>Inverse of square matrix a</td>
</tr>
<tr>
<td>x=np.linalg.solve(a,b)</td>
<td>Solution of ax=b (using pseudo inverse)</td>
</tr>
<tr>
<td>[U,S,V] = np.linalg.svd(a)</td>
<td>Singular value decomposition of a (V is transposed!)</td>
</tr>
<tr>
<td>np.fft.fft2(a), np.fft.ifft2(a)</td>
<td>Inverse 2D fourier transform of a</td>
</tr>
<tr>
<td>np.sqrt(a)</td>
<td>Sets all elements &gt; 0 of a to 5</td>
</tr>
<tr>
<td>arr=arr1.reshape(3,2)</td>
<td>Reshapes arr1 to new size (equal number of items)</td>
</tr>
</tbody>
</table>

## List Comprehension (fast list manipulation)

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[x*y for x in range(10)]</td>
<td>Print all elements of a list</td>
</tr>
<tr>
<td>M = [x*y if x!=None else -1 for x in [0,1,None,2]]</td>
<td>Print a modified list</td>
</tr>
</tbody>
</table>

## Subject and MatLab

- **Data Copying**: MatLab always uses deep copying. b = a -> b and a contain separated data in memory
- **Indexing**: MatLab uses one-based indexing
- **Ranges**: 1:4 means the items at one-based indices 1,2,3,4. Both boundaries are included in the range.

## Python/Numpy

- Python usually creates shallow copies (deep copy only if necessary). Therefore a and b share the same data.
- Python always uses zero-based indexing
- In Python the same is achieved by 0:4 -> [0,1,2,3] The second boundary is always excluded!