Common Syntax Structures

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

Note: Indentation is important for control sequences!

### Assignment

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

### Output

```
print(exp [,expr2…])
```

### Comment

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

### Selection

```
if(boolean_exp):
  stmt
[elif(boolean_exp):
  stmt]
[else:
  stmt]
```

### Repetition

```
while(boolean_exp):
  stmt
```

### Traversal

```
for var in obj:
  stmt
```

### Exception Handling

```
try:
  stmt ...
except [exc_type] [,var]:
  stmt ...
```

### Function

```
def fctname(params):
  """doc-string"
  stmt
  return obj
```

```
ret = fctname(args)  
```

```
#ret is [6,"done"]
```

### Common Data Types

- **int**: Integer (32/64bit)  
  ```
  3, -4, 0
  ```

- **float**: Floating point number  
  ```
  3.0, -6.55, float('nan')
  ```

- **complex**: Complex number  
  ```
  2+3j, 4j, 5-0j
  ```

- **str**: String of characters  
  ```
  "Python"
  ```

- **byte**: Sequence of integers  
  ```
  b"Python"
  ```

- **list**: Mutable sequence  
  ```
  [2,], [2.3,"a"], [2.3,-1]
  ```

- **tuple**: Immutable sequence  
  ```
  (2,), (2.3,"a"), (2.3,-1)
  ```

- **dict**: Mapping, dictionary  
  ```
  D["key"]
  ```

- **Common List (L) and Tuple (T) Methods**
  ```
  LT[idx], LT[idx1:idx2]
  LT.count(obj)  
  LT.index(obj)
  ```

- **Common Dictionary (D) Methods**
  ```
  D["key"]
  D["key"] = obj
  "key" in D
  D.clear()
  D.keys()
  ```

### Operators and their Precedence

- ****
  ```
  Exponentation
  ```

- **\*, /, %**
  ```
  Multiply, Divide, Mod
  ```

- **+, -**
  ```
  Add, Subtract
  ```

- **|, &, ^, ~**
  ```
  Binary And, Or, Xor, Not
  ```

- **>, <, <=, >=, !=, ==**
  ```
  Membership tests [2 in (1,2,3)]->True
  ```

- **in, not in**
  ```
  Boolean operators
  ```

### Common Built-in Functions

- **abs(x)**
  ```
  Absolute value of x
  ```

- **float(x)**
  ```
  Convert x to float / int (if possible)
  ```

- **len(s)**
  ```
  Number of items in sequence (list, tuple, ...)
  ```

- **range(x,y)**
  ```
  A list [x, x+1, x+2, ..., y-1] (y excluded)
  ```

- **dict()**
  ```
  Empty dictionary
  ```

- **list()**
  ```
  Empty list
  ```

- **tuple()**
  ```
  Empty tuple
  ```

### Common Functions of Module `math` (from `math import *`)

- **cos(x)**, **sin(x)**, **tan(x)**
  ```
  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)
  ```

### Module Import

Example: How to call method `plot` of module `itom`

```
import itom
itom.plot(args, …)
```

```
from itom import plot
plot(args, …)
```

```
from itom import *  
```

```
import itom as fct
fct.plot(args, …)
```

```
import itom as fct
fct.plot(args, …)  
```
### Working with dataIO-Devices (Grabber, AD-Converter...)

- **pluginHelp(“name”)**  
  Prints information about plugin
- **dataOI(“name”,params)**  
  Creates obj (instance) of device
- **obj.getParam(“name”)**  
  Returns value of parameter
- **obj.setParam(“name”,val)**  
  Sets parameter to val
- **obj.startDevice()**  
  Starts device (camera...)
- **obj.stopDevice()**  
  Stops device (camera...)
- **obj.acquire()**  
  Triggers image acquisition
- **obj.setPosRel(idx1,pos1,…)**  
  Relatively moves axis idx1 by pos1
- **obj.setPosAbs(idx1,pos1,…)**  
  Moves axis idx1 to pos1
- **obj.copyVal(obj)**  
  Returns shallow copy of array containing the references to last acquired image
- **obj.setAutoGrabbing(bool)**  
  En-/Disables continuous grab for connected live views

#### Example

```python
arr=np.ndarray([2,3],'uint8')
dObj=dataObject([2,3],'uint8')
```

### Numpy.array (import numpy as np, np.array)

- **arr=np.ndarray([2,3],'uint8')**  
  creates a randomly filled 2x3 array with type uint8
- **arr=np.array([[1,2,3],[4,5,6]])**  
  creates the 2x3 array [[1,2,3],[4,5,6]]
- **arr=np.array(dObj, copy=False)**  
  converts np.array <-> dataObject
- **arr=np.array([1,2,3])**  
  sets all values of array to value 7
- **arr=np.zeros([3,4],'float32')**  
  3x4 array filled with zeros of type float32
- **arr=np.eye(3,'float32')**  
  3x3 identity matrix (type: float32)
- **arr= np.ones([3,4],'float32')**  
  3x4 array filled with ones of type float32
- **arr=arr1.reshape([3,2])**  
  reshapes arr1 to new size (equal number of items)

### DataObject

- **obj.getParam(“name”)**  
  Returns value of parameter
- **obj.setParam(“name”,val)**  
  Sets parameter to val
- **obj.startDevice()**  
  Starts device (camera...)
- **obj.stopDevice()**  
  Stops device (camera...)
- **obj.acquire()**  
  Triggers image acquisition
- **obj.setPosRel(idx1,pos1,…)**  
  Relatively moves axis idx1 by pos1
- **obj.setPosAbs(idx1,pos1,…)**  
  Moves axis idx1 to pos1
- **obj.copyVal(obj)**  
  Returns shallow copy of array containing the references to last acquired image
- **obj.setAutoGrabbing(bool)**  
  En-/Disables continuous grab for connected live views

#### Example

```python
arr=np.array([[1,2,3],[4,5,6]])
dObj = dataObject([2,3],data=(1,2,3,4,5,6))
```

### Indexing

- **arr.shape**  
  Returns size tuple ([2,3])
- **arr.ndim**  
  Returns number of dimensions (here: 2)
- **arr.size**  
  Returns size of first dimensions (here: 2nd and 3rd columns)
- **arr[:,:]=7**  
  sets all values of array to value 7
- **arr[0,1] = 7**  
  sets all elements > 0 of array to 5
- **arr[np.isnan(arr)]=5**  
  sets all NaN elements of arr to 5

### Plots

- **plot(dObj)**  
  1D or 2D plot of dObj (depending on its size)
- **livelImage(dataIO-instance)**  
  Live view of camera

### Common DataObject and Numpy.Array Data Types

- **“uint8”, “int8”, “uint16”, “int16”, “uint32”, “int32”, “float32”, “float64”**  
  Floating point numbers
- **“complex64”, “complex128”**  
  Complex values (64 = 2x32 bit)
- **“rgba32”**  
  Color value (type itom.rgba)

### Subject

<table>
<thead>
<tr>
<th>Data Copying</th>
<th>Matlab always uses deep copying. b = a -&gt; b and a contain separate data in memory</th>
<th>Python usually creates shallow copies (deep copy only if necessary). Therefore a and b share the same data.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Indexing</td>
<td>Matlab uses one-based indexing</td>
<td>Python always uses zero-based indexing</td>
</tr>
<tr>
<td>Ranges</td>
<td>In Python the same is achieved by 0:-1 -&gt; [0,1,2,3]</td>
<td>The second boundary is always excluded!</td>
</tr>
</tbody>
</table>