ICAT Python CheatSheet

script-template

To create a script this could be a template:

```
#! python

print 'Hello World'
```

The first line defines the interpreter which will be used by the OS.
After give the execute-right it could be executed:

```
$ chmod +x myScript.py
$ ./myScript.py
Hello World
```

prompt

For testing quickly it could be usefull to get a prompt. Simply type `python` and you will get to it:

```
$ python
Python 2.6.4 |EPD 6.1-1 (32-bit)
[GCC 4.0.1 (Apple Inc. build 5465)] on darwin
Type "help", "copyright", "credits" ...
>>>
```

The >>> are the prompt. If you use multilineconstructs you will get dots to show this:

```
>>> if True:
...     print 'I am True'
I am True
>>>
```

Indentation

The indentation in python is part of the language and defines blocks. The first statement e.g. has to be close left, if you change this you will raise an error.

```
>>>     print 'error'
  File "<stdin>", line 1
    print 'error'
    ^
IndentationError: unexpected indent
```

Types / structures

The types are not fixed, it's possible to reassign a variable with another type

```
>>> x = 1
>>> type(x)
<type 'int'>
>>> x = "Hello"
>>> type(x)
<type 'str'>
```

Strings

To assign a multiline Text:

```
>>> x= """
1st line
2nd line
"""
>>> print x
1st line
2nd line
```

Operators

looking for types by yourself

To assign a value you can simply assign it to a variable. You don't have to worry about the type. You will get a TypeError if you try to execute commands on the variable which are not implemented for this type.

```
>>> 2 - 1
1
>>> 'c' - 'c'
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: unsupported operand type(s) for -:
'str' and 'str'
```

Strings

There are a lot of Operators around, just a view to get an idea:

`+`	concatenate	`"1"+"2"`	`"12"`
`*`	multiply	`"#"*5`	`"#####"`
`len()`	give length	`len("12345")`	`5`
`find()`	index of 1st sight	`"12345".find("2")`	`1`
`center()`	center string	`"1".center(3,"-")`	`-1-`
`rjust()`	align right	`"1".rjust(3,"-")`	`--1`

Control-Structures

if-then-else

The if-then-else construct is simply this:

```
>>> if True:
...     print 'here its true'
... else:
...     print 'now the false part'
```

for

To iterate:

```
>>> for item in [1,2,3]:
...     print item
1
2
3
```

break,continue,pass

To control the for-structure you can use break,continue,pass.

```
for item in [1,2,3,4,5]:
  f item==1:
    pass      # we will do nothing and proceed
  if item==2:
    continue # skip this iteration
  if item==3:
    break    # exits the for-construct
  print '.'  # will be executed after pass
else:
  pass # this else will be executed if no break is rai
```

True has to be substituted by some condition.

conditions

Depending on the type you could use various compare-operators, just a few:

`==`	equal value	`1==1`	`True`
`!=`	not equal value	`1!=1`	`False`
`>=`	greater or equal	`2>=1`	`True`
`<=`	less or equalue	`1<=2`	`True`
`.startswith()`	true when sta...	`"Hel".startswith("H")`	`True`

Librarys

To extend yout functionality import additional packages

import

For exapmple to use sqrt() import the lib math:

```
>>> sqrt(9)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'sqrt' is not defined
>>> import math
>>> math.sqrt(9)
3.0
>>> from math import sqrt
>>> sqrt(9)
3.0
>>> from math import *
>>> ceil(3.6)
4.0
```

Control-Structures

if

OOP

Object

```
class myC(object):
    def __init__(self,a):
        self.val = a
    def __del__(self):
        del self.val
    def __str__(self):
        return "My Value is: '%s'" % self.val
    def func(self,b):
        self.val *= b
```

Child

The child will inheritate everything from the parent. You could redefine function. Due to the polymorphism it depends on the object which function is called.

```
class childC(myC):
    def func(self,b):
        self.val += b
```