

## Leibniz-Rechenzentrum der Bayerischen Akademie der Wissenschaften



## Advanced python programming

Ferdinand.Jamitzky@LRZ.de





- a list is defined by square brackets
- a list comprehension uses square brackets and "for in"
   >> x = [1,2,3,4,5]
   >> y = [ i for i in x]

```
'<br>'.join([s.split('\n') for s in open("file.txt").readlines()])
```

```
out=""
for s in open("file.txt").readlines():
    out = out + s.split('\n')
```



- range(10000) would generate a list of 10000 number although they would later on not be needed.
- generators to the rescue!!
- only generate what you really need
- new keyword: **yield** (instead of **return**)
- >>> def createGenerator():

```
... mylist = range(3)
```

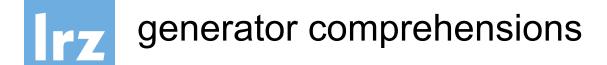
... for i in mylist:

```
.. yield i*i
```

```
• • •
```

>>> a=createGenerator()

```
>>> next(a)
```



- like list comprehensions, but computed only when needed
  >>> a = (i\*\*4 for i in range(8))
  >>> next(a)
  0
  >>> next(a)
  1
  >>> list(a)
  [16, 81]
- >>> import random
- >>> r=random.uniform
- >>> np=100\_000\_000

```
>>> sum((r(0,1)**2+r(0,1)**2 < 1) for i in range(np))/np*4.
3.141244</pre>
```





## dictionaries aka associative arrays aka key/value stores

dictionary comprehensions:

>>> {i:i\*\*2 for i in range(4)}

- $\{0: 0, 1: 1, 2: 4, 3: 9\}$
- >>> a.keys()
- >>> a.values()



- function names with leading and trailing underscores are special in python ("magic methods")
- >>> print(a)

is translated to:

- >>> a.\_\_print\_\_()
  and
- >>> a+b
- >>> a.\_\_add\_\_(b)
  >>> f(x)
- >>> f.\_\_call\_\_(x)





using try you can catch an exception that would normally stop the program

```
x=range(10)
y=[0]*10
for i in range(10):
    try:
        y[i]=1./x[i]
    except:
        y[i]=0.
```





decorators are syntactic sugar for applying a function
and overwriting it.
@mydecorator
def myfunc():
 pass
is the same as:

```
def myfunc():
    pass
myfunc = mydecorator(myfunc)
```





The with statement allows for different contexts with EXPR as VAR: BLOCK

roughly translates into this:

VAR = EXPR
VAR.\_\_enter\_\_()
try:
 BLOCK
finally:
 VAR.\_\_exit\_\_()



You need a context manager (has enter and exit methods) Examples:

• opening and automatically closing a file

```
with open("/etc/passwd") as f:
    df=f.readlines()
```

- database transactions
- temporary option settings
- ThreadPoolExecutor
- log file on/off
- cd to a different folder and back
- set debug verbose level
- change the output format or output destination

```
with redirect_stdout(sys.stderr):
    help(pow)
```



- AOP is about separating out *Aspects*
- You can switch contexts (like log-file on/off)

```
from contextlib import contextmanager
@contextmanager
def tag(name):
    print("<%s>" % name)
    yield
    print("</%s>" % name)
>>> with tag("h1"):
... print("foo")
```

```
<h1>foo</h1>
```



- better "if then else" block
- wildcard \_
- combine patterns with |

```
match status:
    case 400:
        print("Bad request")
    case 401 | 403 | 404:
        print("not found")
    case _:
        print("something is wrong with the internets")
```



```
async def ticker(delay,to):
    for i in range(to):
        yield i
        await asyncio.sleep(delay)
```

defines an asynchronous function, which waits for delay. It can be called in the following way:

```
async for i in ticker(1,10):
    print(f'tick {i}')
```