cap_sage_technique

Fallunterscheidung

```
def f1(x):
    if x > 0: return 1
    else: return 0
def g1(x):
    x = 1 if x >0 else 0; return x
```

```
print f1(2), g1(2)
```

```
1 1
```

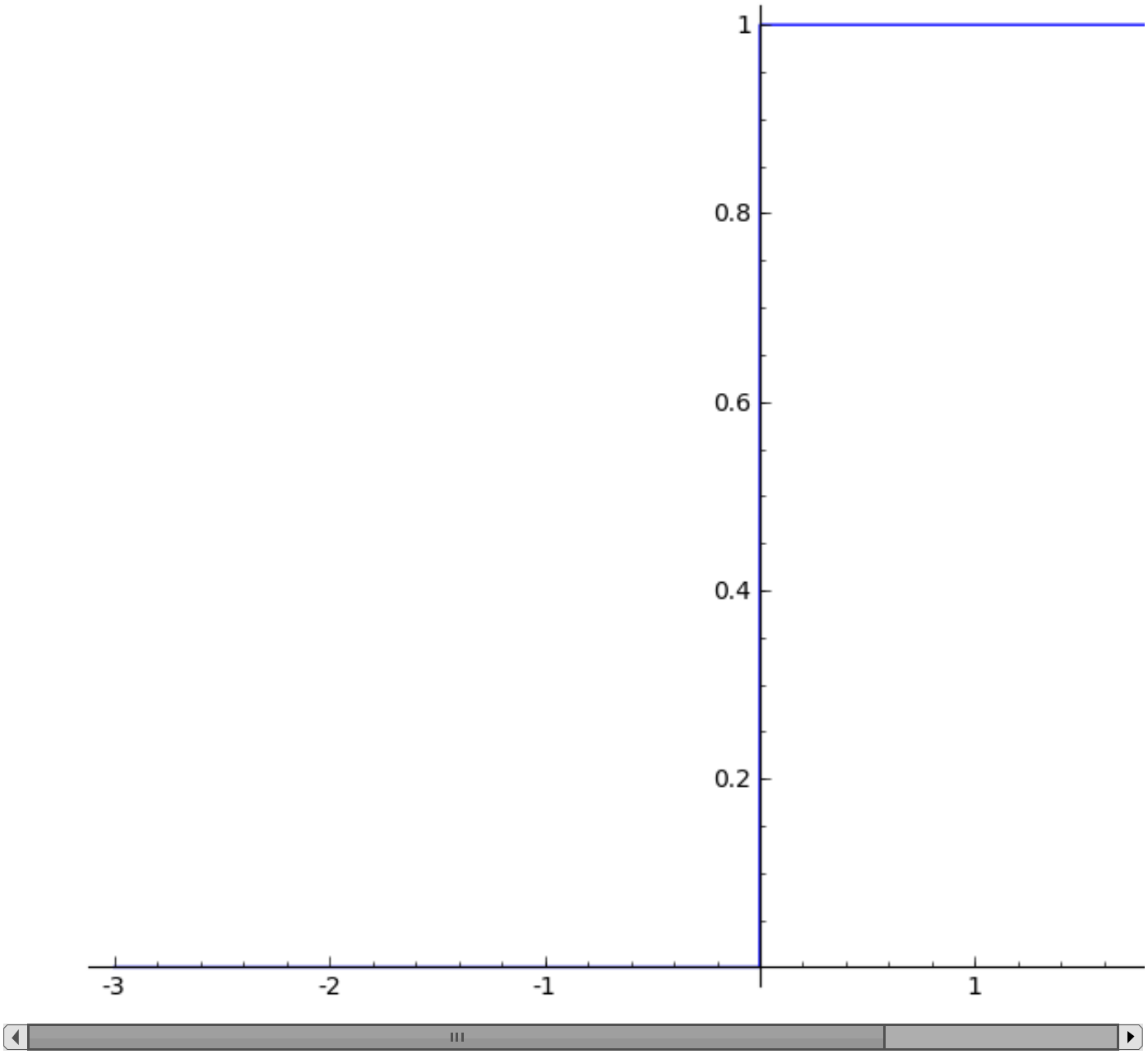
```
print f1(-3), g1(-3)
```

```
0 0
```

```
print f1(x),g1(x)
```

```
0 0
```

```
plot(g1, (-3,3))
```



Die Fakultät:

iterative Verfahren

```
def fac1(n):
    res = 1
    for k in range(1,n+1):
        res = res * k
    return res
```

```
fac1(2000)
```

```
331627509245063324117539338057632403828111720810578039457193
807790560082240027323085973259225540235294122583410925808481
```

961313866335263436889056340585561639406051172525718706478563940524395746703767410872297043468415834375243158087753364512746859247408032408946561507233250652797655757179671536718689359158716017172326571561100042140124204338425737127001758835477928352899666585340557985490365736635013338655040117201215263548152152246920995206031564418565480675946497051552288205234899508140655366789695321014676226713320268315522051944944616182302652972263150257475204829606475092739416585628353177957448286450373991327334177263608852490093506621610144459709412707821638315723020199499149583164709427744738703279855496742986088382415247883438746959582925774057453983750158581546813629421799813599481016556563876034227312912250384709872909626622461971315502018951355831653578714922909167790497022470946119376077868443225590564873626653037738465039078804952460071254940261454136302754913671583406097831074945282217490781347709693241556280513586006905946199652573107411770815199225645167785714580565476095237746301667942248844448579834980154803262082989096581888619376692828279888453584639896594213952984465291092009103494499158285880507618679249463851808798745128914080193400746209872957859964365065589561241023101869055606030878362911050568998383410799367902052076858669183477906558544700148692656924376124280974200671728463619392496986284687199934503938893672717273456170035486747750910295552395354794110742191330135681951462766417542161587625262858089801222443890248677182054959415012717675717874958616196659318788551418357820926014820717773303430496908207058995870138198081303559016076290838857456128826182483576739218303118414719133986892842344000779246691209766334944374732356365720488444783318549416930301245316762327453684744738244850922831399525097325059791270310476836014811911022269769382367005756561240029057604385285290293760647953345817939605262549107186663869354766108455046198102084050635827676523932495196859541716724193295306836734955440045863598381610430662753060542358075589410827888042782595108988063541056791795080688782869810219010900148352061688883720250310665922068601485327820882635365580436056867812841692171330471411763121758957758475312351723099054982921013468730420589801441806387538266404237759406280877253702265426530580862379301422675821187143506363403001732518182620760397473695952026426323641454468511134045838385101013694131303485622191663162389263276581535501127659969158824533457435437863683173730673296589355199694458236872786577008797498899923435555662406828347637846851838449736488510322422211056120129582965719136810869382547576411888687934646192151144738836269591643672490071653428228152661247800463921703637236279407577845420910483054616561906221742869816029733020199281385488268195100728286970107073750092766648750217477551508748246720274170031581122805896178122160747437947510950626745812525183766821577128078614992558761323529504223463878789576446613629039412766597804420209228133798711590089626487894254925003566670632909441579372986743421470507213588932019580 72

```
4984295225955890127548239717733257229103257609297907332995450
2640474650245080809469116072632087494143973000704111418595530
5765481918200244969776111346318195282761590964189790958117333
0889104329452449785351470141124421430554860896395783783473253
3291438925288393986256273242862775563140463830389168421633113
0957196597846633855149231619633567535513840342580416291983782
5217701531753387302846108418865541383291719513321178957285416
3682817932512931237521541926970269703299477643823386483008871
0566638386829408848773072176226884902308493466119426018027261
0050782157410060548482013478595781027707077806555127725405016
6066253216415004808772403047611929032210154385353138685538486
9079534117651957118868373988068389579274374968349814292329219
0901439368436553333593078201813129934550242060445633405786069
1505603394899523321800434359967256623927196435402872055475012
3197067479731312681352365374408566226320676883758513278289625
3418129776246970795434360034923431592396747636389121152854066
6213911247447051255226342701239527018127045491648045932248108
0095230679317596775558101167994000524980630376314134441226903
3557999160092592480750524855415682662817608154463083054066774
4441864204108373119093130001154470560277773724378067188899770
2727678124719883285769584421758889516046786820481001004781646
8385324881342708340798684866321627202088233087278190853788454
6021728873121907393965209260229101477527080930865364979858554
5027928981460368843182150863724621696787228216934737059928627
6909209029883201668301702734202597656717098633112163495021712
7119650264054228231759630874475301847194095524263411498469508
8000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000
000000000000000000000000
```

```
def fac2(n):
    k = 1
    res = 1
    while(k <= n):
        res = res*k; k += 1
    return res
```

```
fac2(100)
```

```
9332621544394415268169923885626670049071596826438162146859296
5999932299156089414639761565182862536979208272237582511852109
0000000000000000000000
```

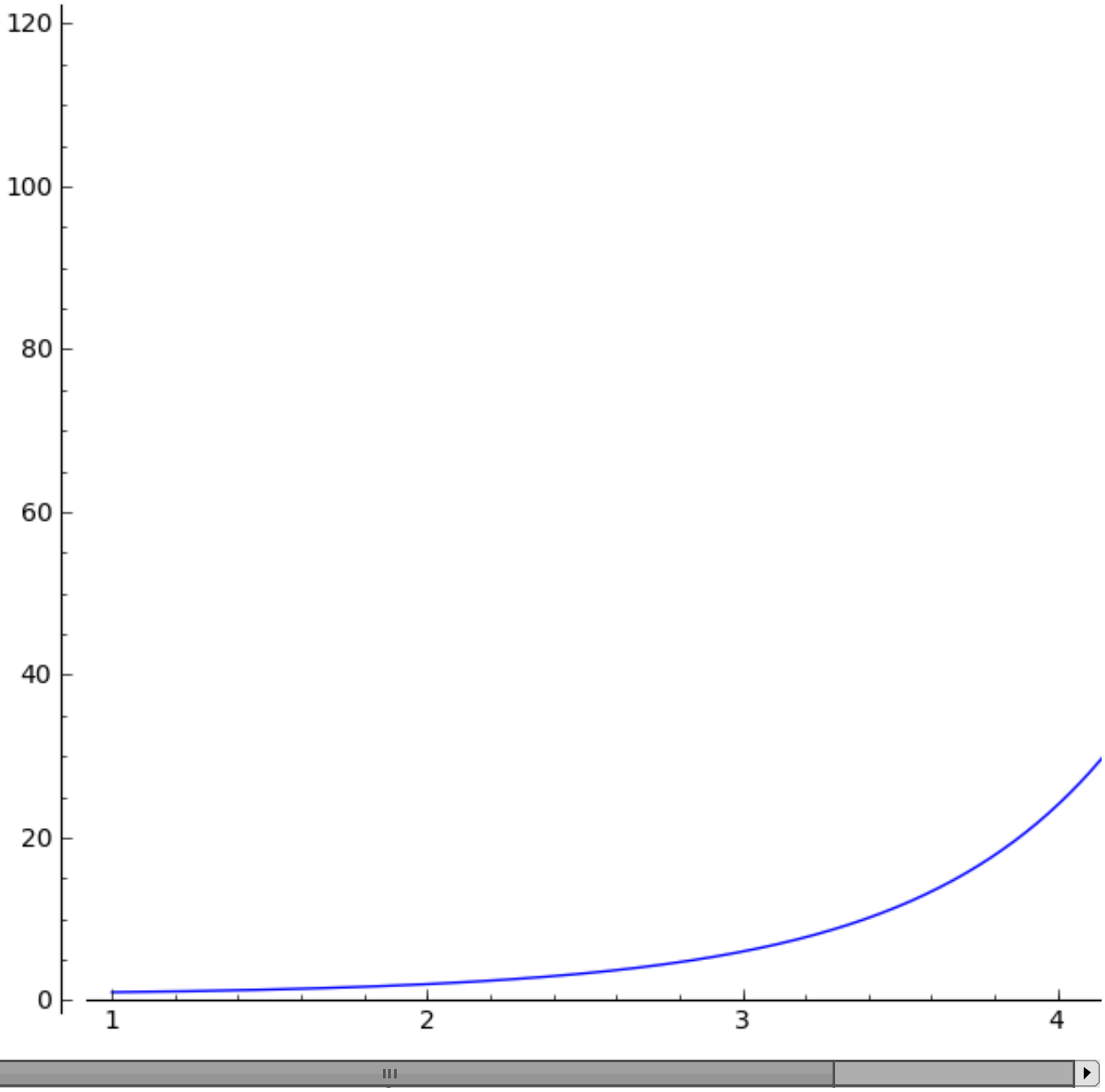
```
prod(range(1,101))
```

```
933262154439441526816992388562667004907159682643816214685929659999322991560894146397615651828625369792082722375825118521090000000000000000000000L
```

```
factorial(100)
```

```
933262154439441526816992388562667004907159682643816214685929659999322991560894146397615651828625369792082722375825118521090000000000000000000000
```

```
plot(factorial,(1,5))
```



Die Fakultät: rekursiv

```
def fac3(n):
    if n == 0: return 1
```

```
    return n*fac3(n-1)
```

```
fac3(500)
```

```
1220136825991110068701238785423046926253574342803192842192413
4537315388199760549644750220328186301361647714820358416337872
2004807852051593292854779075719393306037729608590862704291745
4912726344305670173270769461062802310452644218878789465754777
9436778103764427403382736539747138647787849543848959553753799
0612713269843277457155463099772027810145610811883737095310163
2987029563896628911658974769572087926928871281780070265174507
1962439039432253642260523494585012991857150124870696156814162
6934238130088562492468915641267756544818865065938479517753608
5238940335798476363944905313062323749066445048824665075946735
3792518420045936969298102226397195259719094521782333175693458
3328207628200234026269078983424517120062077146409794561161276
1237229913340169552363850942885592018727433795173014586357570
8015873543276888868012039988238470215146760544540766353598417
1289383138968816394874696588175045069263653381750554781286400
0000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000
```

```
fac3(4)
```

```
24
```

```
fac3(100)
```

```
9332621544394415268169923885626670049071596826438162146859296
5999932299156089414639761565182862536979208272237582511852109
0000000000000000000000
```

Fibonaccizahlen

```
def fib1(n):
    if n == 0: return 0
    if n == 1: return 1
    return fib1(n-1)+fib1(n-2)
```

```
fib1(5)
```

```
5
```

```
fib1(30)
```

```
832040
```

```
fibonacci(30)
```

```
832040
```

```
timeit('fib1(30)')
```

```
5 loops, best of 3: 757 ms per loop
```

```
timeit('fibonacci(30)')
```

```
625 loops, best of 3: 7.14 µs per loop
```

```
memo = {0:0, 1:1}
def fib2(n):
    if not n in memo: memo[n] = fib2(n-1) + fib2(n-2)
    return memo[n]
```

```
timeit('fib2(30)')
```

```
625 loops, best of 3: 1.59 µs per loop
```

```
fib2(1000)
```

```
434665576869374564356885276750406258025646605173717804024817255541794905189040387984007925516929592259308032263477520968963322471161642996440906533187938298969649928516003704476137795 28875
```

Schnelles Potenzieren

divide & conquer

```
def potenz(a,n):
    if n == 0: return 1
    if n%2==0: return potenz(a,n/2)**2
    else: return a*potenz(a,n-1)
```

```
potenz(3,16)
```

```
43046721
```

```
3**16
```

```
43046721
```

modulare iterative Implementierung

Zahlen zu einer anderen Basis; iterativ

```
def ziffern1(n,b):
    #least significant first
    q = n; liste = []
    while q > 0:
        liste.append(q%b)
        q = q.quo_rem(b)[0]
    return liste
```

Zahlen zu einer anderen Basis; rekursiv

```
def ziffern2(n,b):
    #least significant first
    if n < b: return [n]
    return [n%b]+ziffern2(n.quo_rem(b)[0],b)
```

```
11.digits(2)
```
```
[1, 1, 0, 1]
```
```
ziffern1(11,2)
```
```
[1, 1, 0, 1]
```
```
ziffern2(11,2)
```
```
[1, 1, 0, 1]
```
```
ziffern1(1023,2)
```
```
[1, 1, 1, 1, 1, 1, 1, 1, 1, 1]
```
```
ziffern2(1023,2)
```
```
[1, 1, 1, 1, 1, 1, 1, 1, 1, 1]
```

iteratives schnelles Potenzieren

```
def itPowerMod(a,n,p):
    digits = ziffern1(n,2)
    result = 1; power = a
    for digit in digits:
        if digit == 1:
            result = (result*power)%p
            power = (power**2)%p
        else: power = (power**2)%p
    return result
```

```
itPowerMod(2,3,5)
```
```
3
```
```
power_mod(2,3,5)
```
```
3
```
```
a = ZZ.random_element(10**10000)
n = ZZ.random_element(10**10000)
p = ZZ.random_element(10**10000)
```

```
timeit('itPowerMod(a,n,p)')
```

```
5 loops, best of 3: 12 s per loop
```

```
timeit('power_mod(a,n,p)')
```

```
5 loops, best of 3: 12 s per loop
```

Erweiterter Euklidscher Algorithmus

```
xgcd(1234,56789)
```

```
(1, -24989, 543)
```

Chinesischer Restsatz

```
crt([2,3,1],[3,4,7])
```

```
71
```

```
crt(range(1,11),[next_prime(k) for k in range(10)])
```

```
Traceback (click to the left of this block for traceback)
...
ValueError: No solution to crt problem since gcd(2,2) does no
divide 1-2
```

Kleiner Satz von Fermat

```
n = ZZ.random_element(10**100)
p = ZZ.random_element(10**100)
```

```
power_mod(n,p-1,p)
```

```
28808554804449066151094195640359649705529691431974683069394666
60973714870480046970367102383075
```

```
p = next_prime(p)
```

```
power_mod(n,p-1,p)
```

```
1
```

```
p.is_prime()
```

```
True
```

```
```