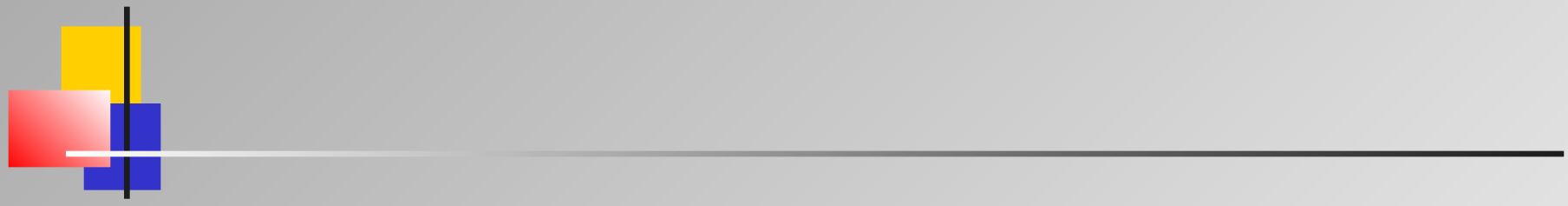
AA 2020-21

DMI - Università degli Studi di Cagliari

OOP and Scripting in Python

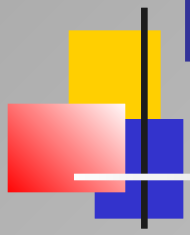
Part 1 - Basic Features

Giuliano Armano – DMI Univ. di Cagliari

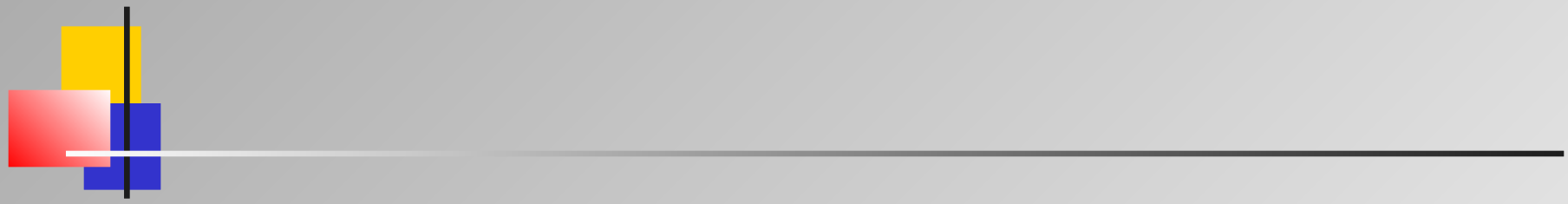


Part 1 - Basic Features

Python: Basic Features



- Data Structures
- Control Structures
- Functions
- Polymorphic variables
- Garbage Collection



Data Structures

Part 1 - Basic Features: Data Structures

Data Structures (conventional)

- Numbers (int, long, double) YES
- Booleans YES
- Characters NO
- Strings YES
- Arrays NO
- Records NO

Numbers

```
>>> x = 1              # int
>>> print(x)
1
>>> z = x + 2.3        # double
>>> print(z)
3.3
>>> print(x+z)
4.3
```

Booleans

```
>>> print(True or False)
True
>>> print(True and False)
False
>>>
```

Strings

Strings are immutable

```
>>> print("Hello, world!")
Hello, world!
>>> print('Hello, world!')
Hello, world!
>>> a = "blob"
>>> b = a
>>> a += '-one'
>>> print(a,b)
blob-one blob
>>>
```

Data Structures (advanced)

- Big numbers YES
- Complex numbers YES
- Lists YES
- Tuples YES
- Dictionaries YES

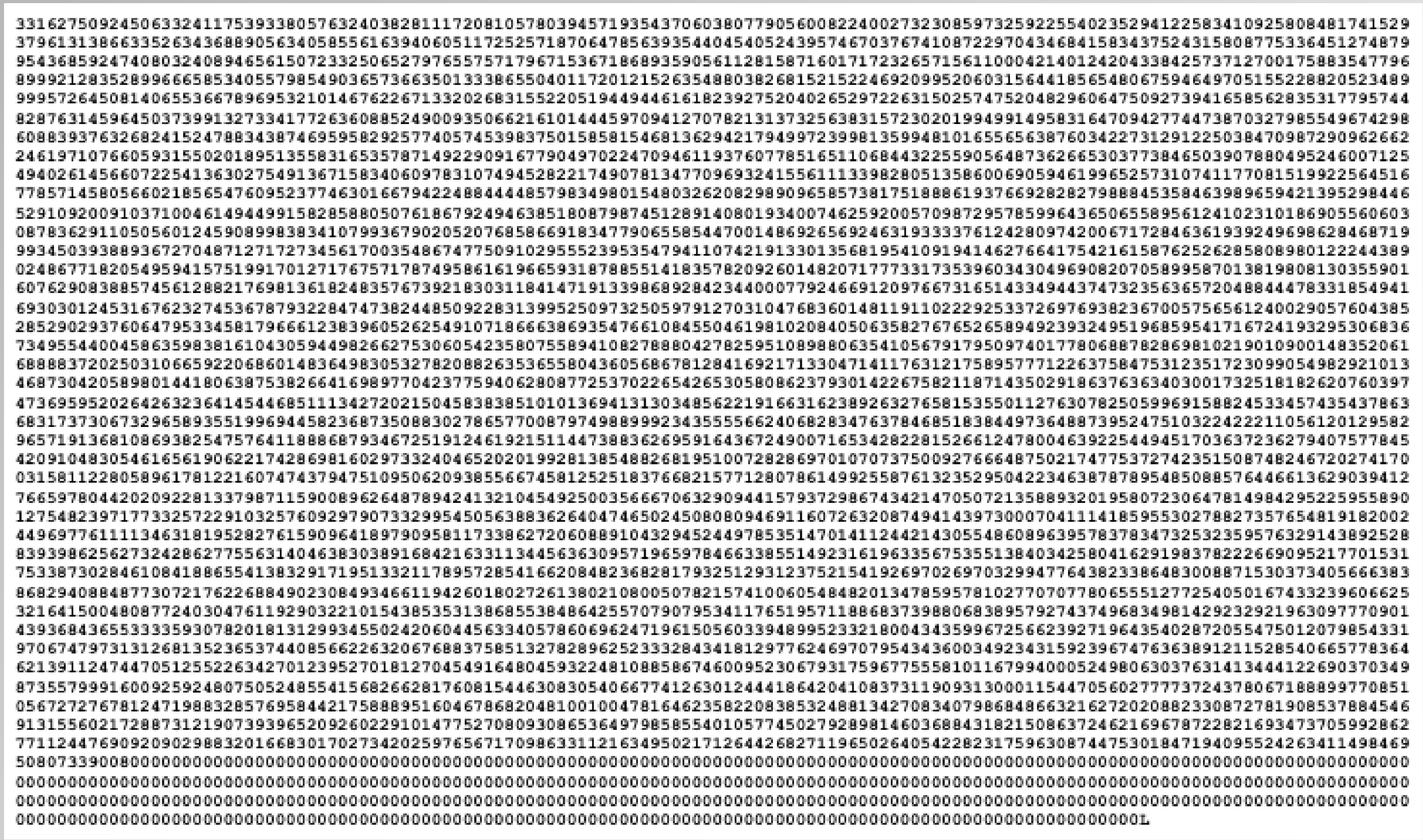
Big Numbers

```
>>> def fact(x):
...     if x <= 1: return 1
...     else: return x * fact(x-1)
...

>>> fact(200)
788657867364790503552363213932185062295135977687173263294742533244359449963403342920304284011984623904177212138919638830257642790242637105061926624952829931113462857270763317237396988943922445621451664240254033291864131227428294853277524242407573903240321257405579568660226031904170324062351700858796178922227896237038973747200000000000000000000000000000000000000000L

>>>
```

Big Numbers

```
>>> def fact(n):
        if n <= 1: return 1
        else: return n * fact(n-1)

>>> import sys
>>> sys.getrecursionlimit()
1000
>>> sys.setrecursionlimit(2500)
>>> # guess what happens now ...
>>> fact(2000)
```

Big Numbers



Complex Numbers

```
>>> c = complex(1.,2.)
>>> c.real
1.0
>>> c.imag
2.0
>>> c
(1+2j)
>>> c += (3+4j)
>>> c
(4+6j)
>>>
```

Lists

```
>>> L = ['a','b','c']
>>> L
['a', 'b', 'c']
>>> L += [1,2]
>>> L
['a', 'b', 'c', 1, 2]
>>> L[2]
'c'
>>> L[-1]
2
```

```
>>> list('blob')
['b', 'l', 'o', 'b']
>>> list('blob')[2]
'o'
>>> list('blob'+'s')
['b', 'l', 'o', 'b', 's']
>>>
```

Tuples

Tuples are immutable

```
>>> T = ('x','y','z')
>>> T
('x', 'y', 'z')
>>> T[0]
'x'
>>> T[0:2]
('x', 'y')
>>> T[2]
'z'
>>> T[-2]
'y'
>>>
```

```
L = list(T)
>>> L += ['w']
>>> L
['x', 'y', 'z', 'w']
>>> T1 = tuple(L)
>>> T
('x', 'y', 'z')
>>> T1
('x', 'y', 'z', 'w')
>>>
```

Tuples

Tuples are immutable

```
>>> a = (1,2)
>>> x, y = a
>>> x
1
>>> y
2
>>>
```

```
>>> b =('a','b',100,20,11)
>>> b
('a', 'b', 100, 20, 11)
>>> b[1]
'b'
>>> b[2:3]
(100,)
>>> b[2:]
(100, 20, 11)
>>>
```

Dictionaries

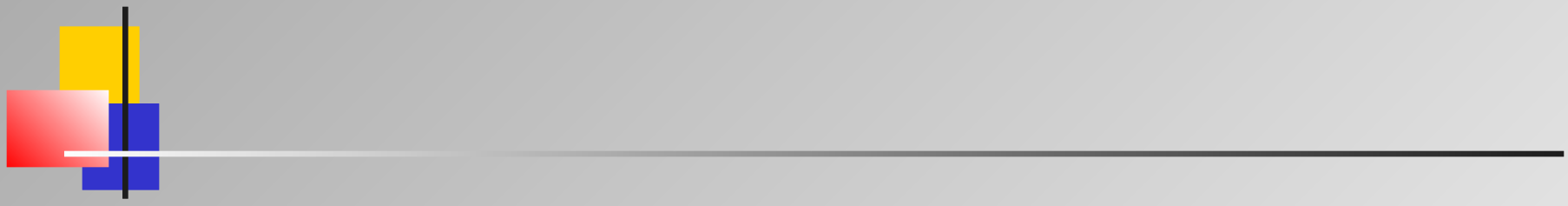
```
>>> D = {'john' : 32, 'paul' : 44, 'helen' : 38}
>>> D
{'paul': 44, 'john': 32, 'helen': 38}
>>> D['john']
32
>>> D['helen']
38
>>> D['helen'] = 37
>>> D
{'paul': 44, 'john': 32, 'helen': 37}
```

Dictionaries

```
>>> D
{'paul': 44, 'john': 32, 'helen': 38}
>>> D.items()
dict_items([('paul', 44), ('john', 32), ('helen', 38)])
>>> D.keys()
dict_keys(['paul', 'john', 'helen'])
>>> D.values()
dict_values([44, 32, 38])
```

Dictionaries

```
>>> D
{'paul': 44, 'john': 32, 'helen': 38}
>>> D['helen']
38
>>> del D['helen']
>>> 'helen' in D
False
>>> D['helen']

Traceback (most recent call last):
  File "<pyshell#96>", line 1, in <module>
    D['helen']
KeyError: 'helen'
>>>
```



Control Structures

Part 1 - Basic Features: Control Structures

Control Structures

- if_then / if_then_else YES
- case (using if_then_elif_else) YES
- while_do YES
- for (very powerful) YES
- Omega-N (using while_do + break) YES

if_then / if_then_else

```
>>> x = 30
>>> if x < 100: print('hello')       # if_then
...

hello
>>>
>>> x = 40
>>> if x == 40: print('foo')      # if_then_else
... else: print('oof')
...

foo
>>>
```

Case

```
>>> y = 3
>>> if y==1: print('one')
... elif y==2: print('two')
... elif y==3: print('three')
... else: print('unknown')
...

three
>>>
```

While_Do

```
>>> x = 5
>>> while x > 0:
...     print('foo',x)
...     x -= 1
...

foo 5
foo 4
foo 3
foo 2
foo 1
>>>
```

For (on an integer range)

```
>>> range(5)
range(0, 5)
>>> list(range(0,5))
[0, 1, 2, 3, 4]
>>> for x in range(5):
...   print('foo',x)
...

foo 0
foo 1
foo 2
foo 3
foo 4
>>>
```

For (on a string)

```
>>> for x in 'blob':
...   print('foo',x)
...

foo b
foo l
foo o
foo b
>>>
```

For (on a list)

```
>>> for x in ['a','b','c','d']:
...   print('foo',x)
...

foo a
foo b
foo c
foo d
>>>
```

For (from string to list and v.v.)

```
>>> x = []
>>> for y in 'abcd':
...     x += y

>>> x
['a', 'b', 'c', 'd']
>>>
>>> xx = ''
>>> for y in x:
...     xx += y
...

>>> xx
'abcd'
```

For (on a tuple)

```
>>> for x in ('a','b','c','d'):
...   print('foo',x)
...

foo a
foo b
foo c
foo d
>>>
```

For (on a dictionary)

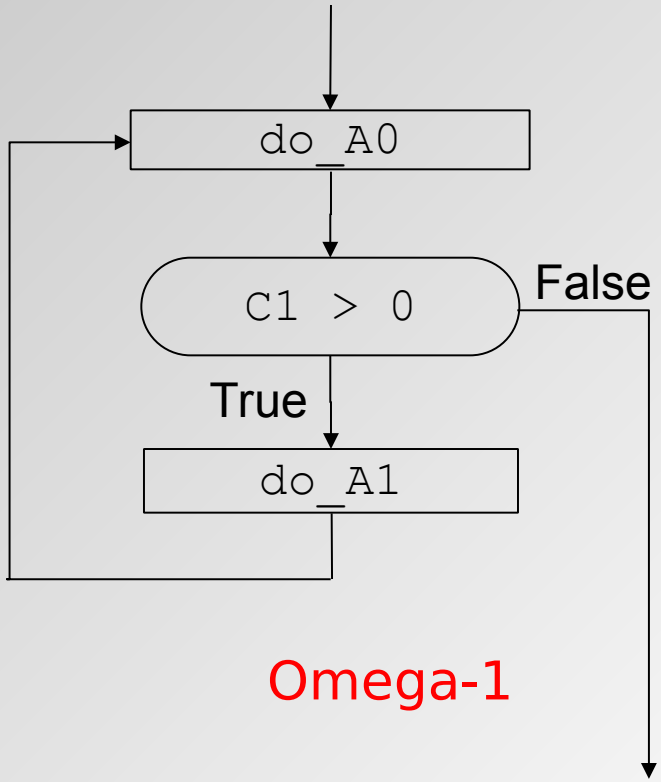
```
>>> d = {'a':1, 'b':2,'c':3}
>>>
>>> for x,y in d.items():
...   print('foo',x,y)
...

foo a 1
foo c 3
foo b 2
>>>
```

For (on a dictionary)

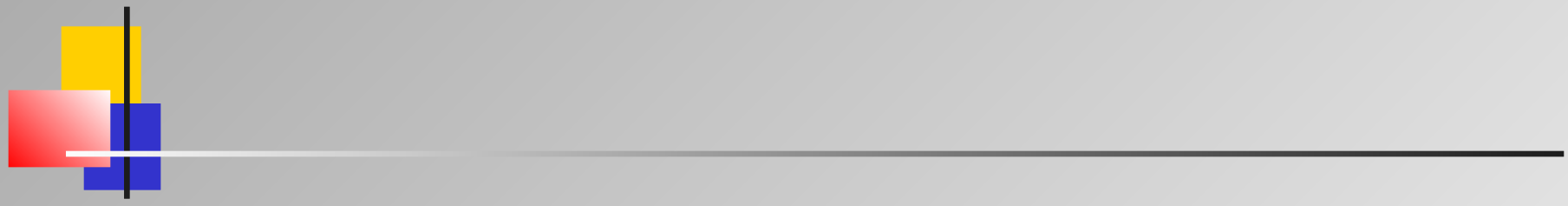
```
>>> d = {'a':1, 'b':2,'c':3}
>>> for k in d.keys():
...     print('keyword = {}, value = {}'.format(k,d[k]))


keyword =  a, value =  1
keyword =  b, value =  2
keyword =  c, value =  3
>>>
```

Omega-N

```
>>> C1 = 2
>>> while True:  # omega-1
...     print('doing A0')
...     if not C1 > 0: break
...     print('doing A1')
...     C1 -= 1
...

doing A0
doing A1
doing A0
doing A1
doing A0
>>>
```



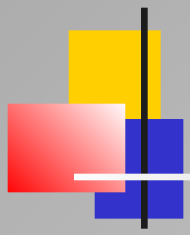
Omega-1



Functions

Part 1 - Basic Features: Functions

Functions



- Arguments' type checking — NO
- Default parameters — YES
- Keyworded parameters — YES
- Apply functions to arglists — YES

Arguments' Type Checking

```
>>> def foo(x):
...   print(x)
...

>>> foo('blob')
blob
>>> foo(34)
34
```

Default parameters

```
>>> def foo(x=0):
...     print(x)
...

>>> foo(10)
10
>>> foo()
0
```

Default parameters

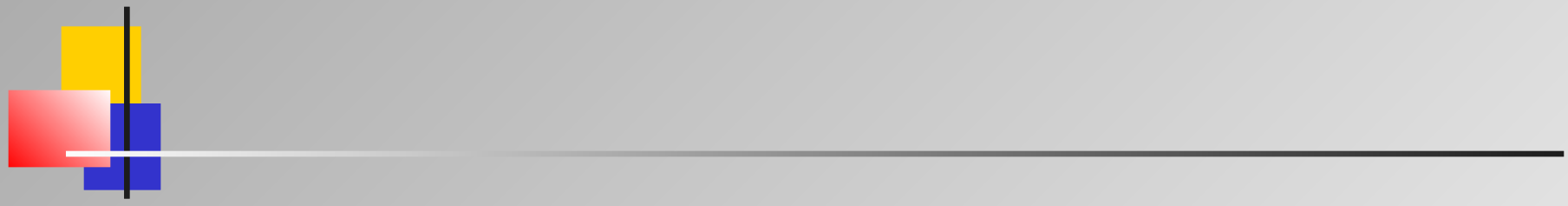
```
>>> x1 = 0
>>> def foo(x=x1):
...   print(x)
...

>>> foo(10)
10
>>> foo()
0
>>> x1 = 33
>>> foo()     # 0 or 33 ? 0
0
>>>
```

Keyworded parameters

```
>>> def foo(x=0,y=0,z=0):
...    print("x,y,z = ",x,y,z)
...

>>> foo()
x,y,z =  0 0 0
>>> foo(11)
x,y,z =  11 0 0
>>> foo(y=33)
x,y,z =  0 33 0
>>> foo(z=1,y=33)
x,y,z =  0 33 1
>>>
```

Apply functions to arglists

```
>>> def foo(x=0,y=0,z=0):
...    print("x={}, y={}, z={}".format(x,y,z))
...

>>> apply(foo,[1,2,3])     # deprecated
x,y,z =  1 2 3
>>> foo(*[1,2,3])          # OK, new programming style
x,y,z =  1 2 3
>>> foo(*[10,20])          # z=0 (default value)
x,y,z =  10 20 0
>>> foo(**{'x':1, 'z':3}) # y=0 (default value)
x,y,z =  1 0 3
>>>
```



Polymorphic Variables

Part 1 - Basic Features: Polymorphic Variables

Polymorphic Variables

- Type declaration (for variables) NO
- Type checking (on assignment) NO
- Scope (lexical vs. dynamic) DYNAMIC
- Extent(automatic vs. dynamic) DYNAMIC

Polymorphic Variables

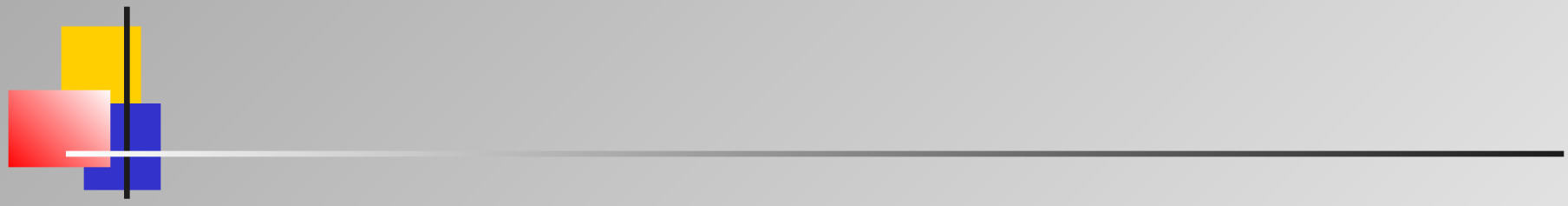
Type Declaration and Type Checking

```
>>> def foo(x):
...   print(x)
...

>>> a = 1
>>> a
1
>>> a = "blob"
>>> a
blob
>>> a = foo
>>> a
<function foo at 0x00A9E170>
```

Polymorphic Variables

Dynamic Scope and Extent

```
>>> x = 35
>>> def foo():
...   print(x)
...

>>> foo()
35
>>> del x
>>> foo()
Traceback (most recent call last):
  File "<pyshell#71>", line 1, in <module> foo()
  File "<pyshell#68>", line 2, in foo print x
NameError: global name 'x' is not defined
```



Garbage Collection

Part 1 - Basic Features: Garbage Collection

Garbage Collection

- No need for object destructors
- Unused values / objects are collected and removed by a suitable algorithm (= distributed garbage collector)