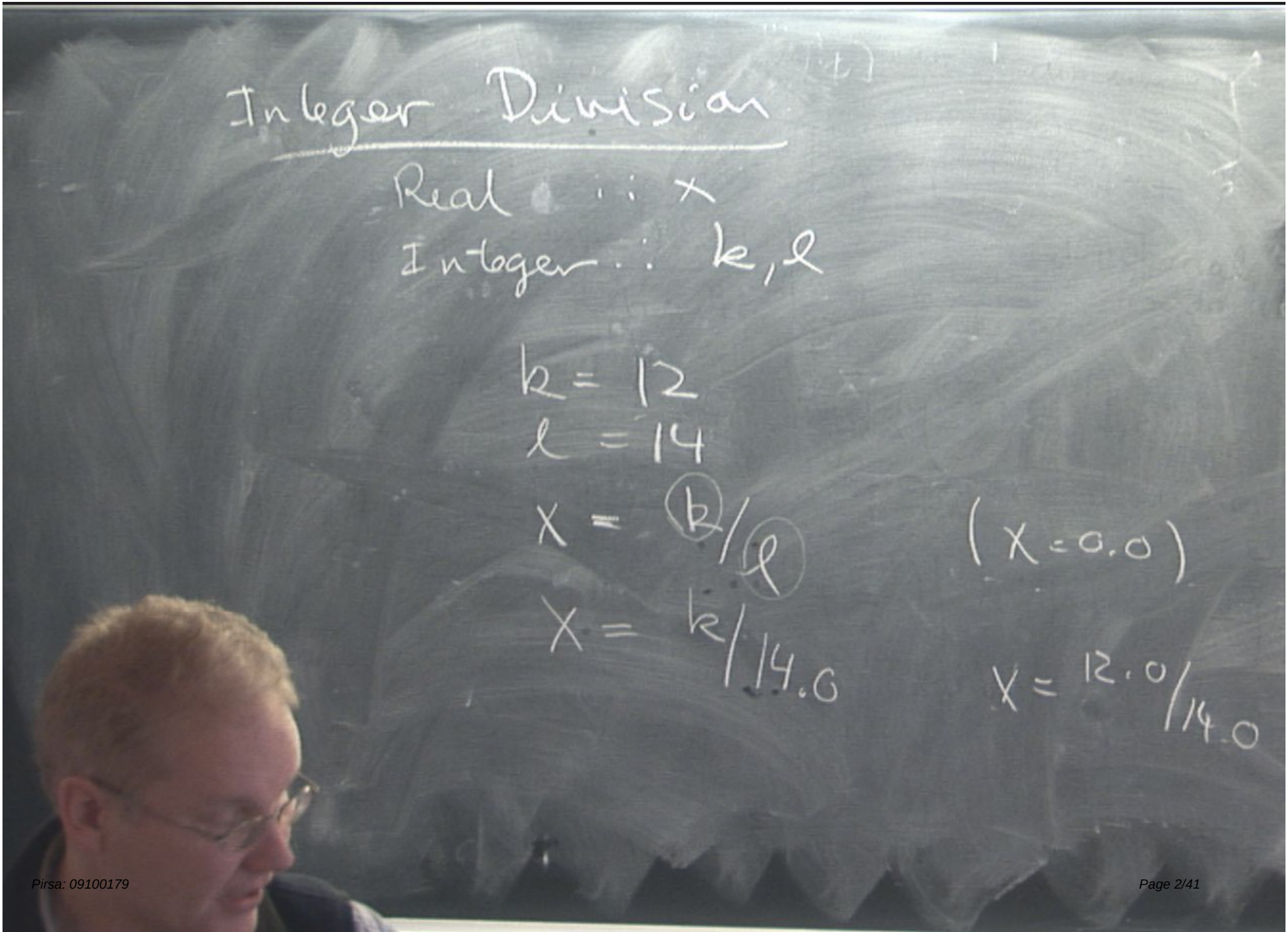
Title: Scientific Computation (PHYS 608) - Lecture 3

Date: Oct 28, 2009 10:30 AM

URL: http://pirsa.org/09100179

Abstract:


Integer Division
Real :: x
Integer :: k, l
k = 12
l = 14
x = k/l
(x = 0.0)
x = k/14.0
x = 12.0/14.0

Integer Division

Real :: x

Integer :: k, l

k = 12

l = 14

x = k/(l*1.0) (x = 0.0)

x = k/14.0

x = 12.0/14.0

Optional Arguments

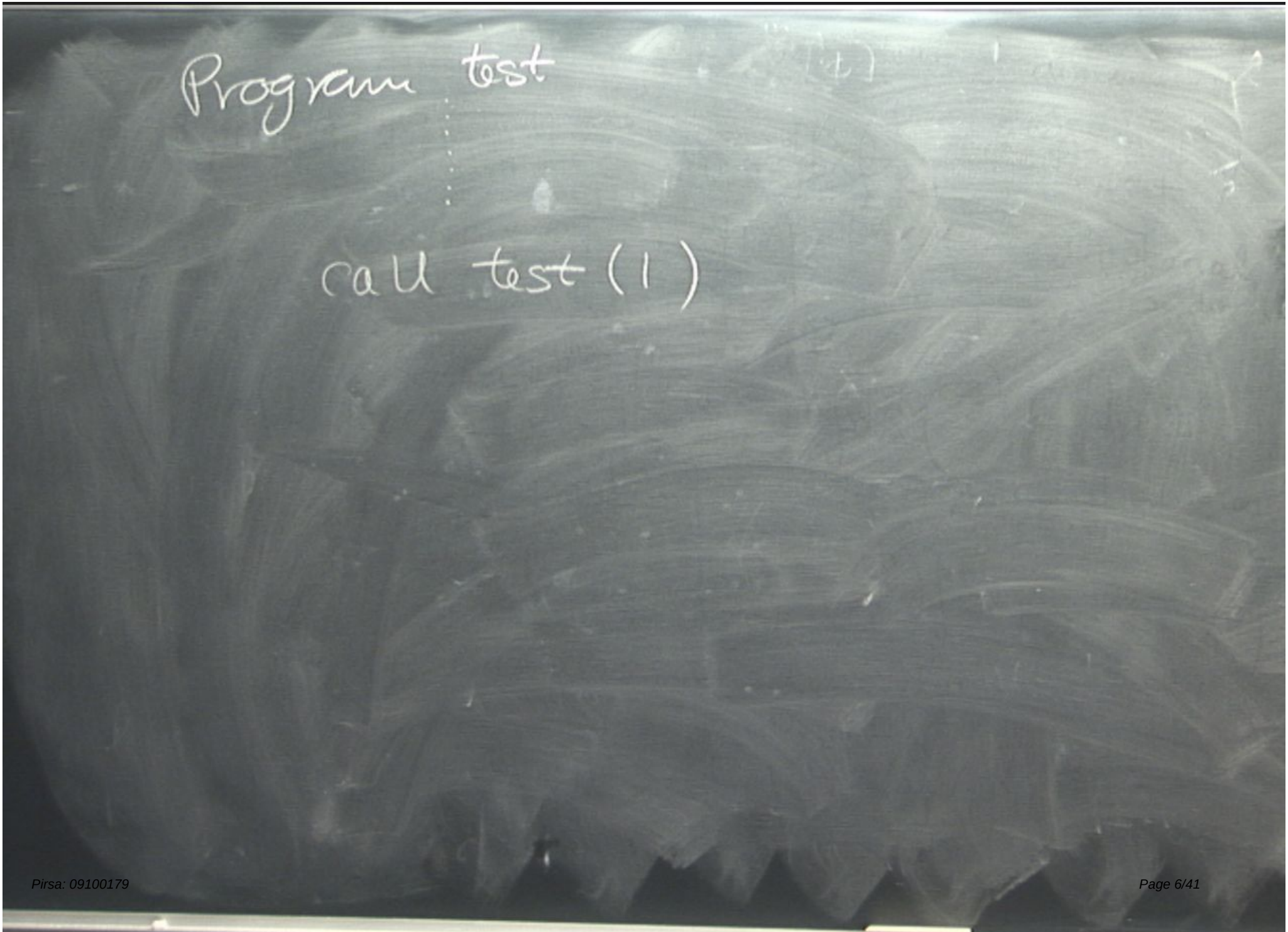
If we declare an explicit face

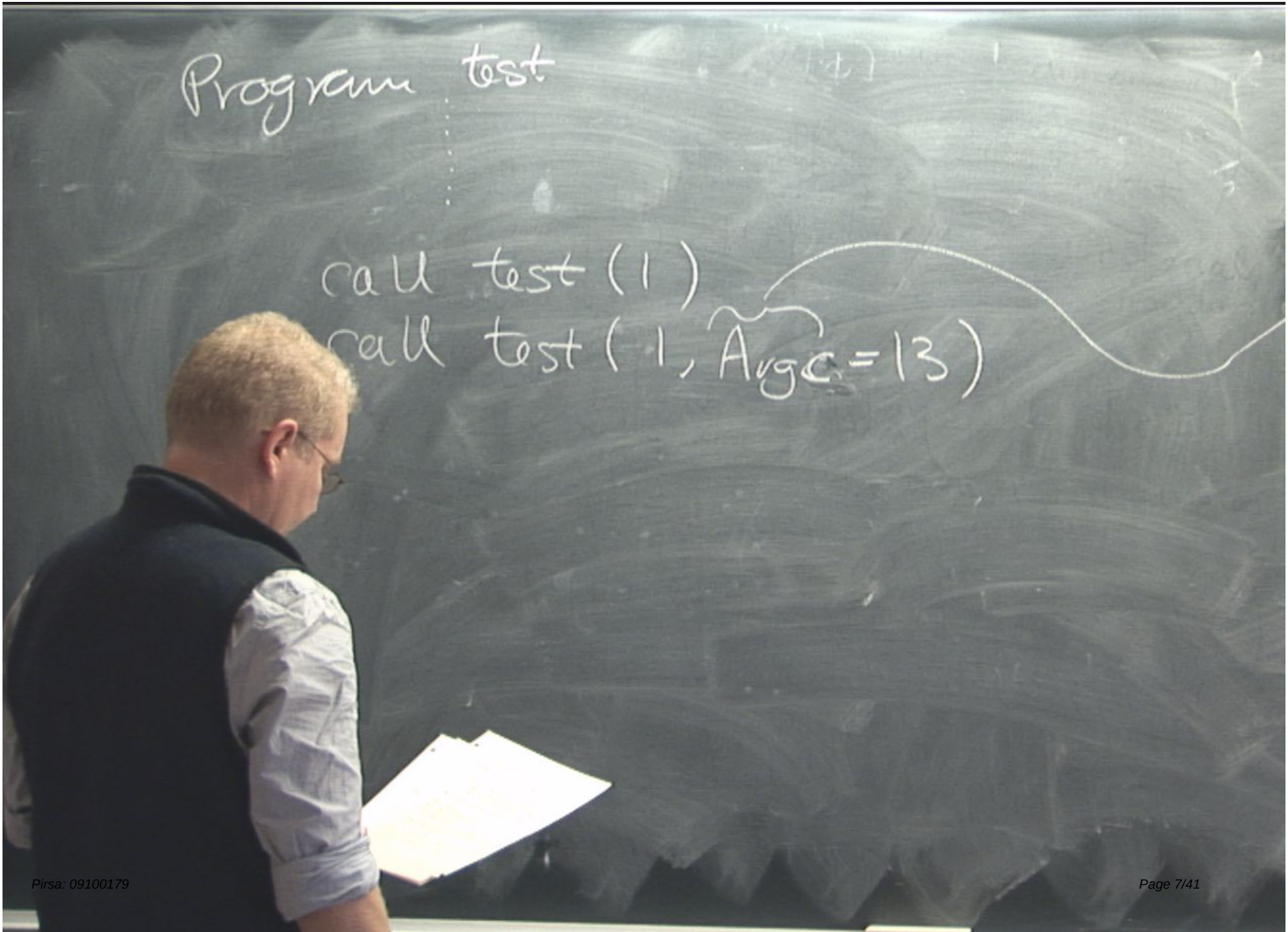
```
Interface
  subroutine test(Arga, Argb, Argc)
    integer, intent(inout) :: Arga
```

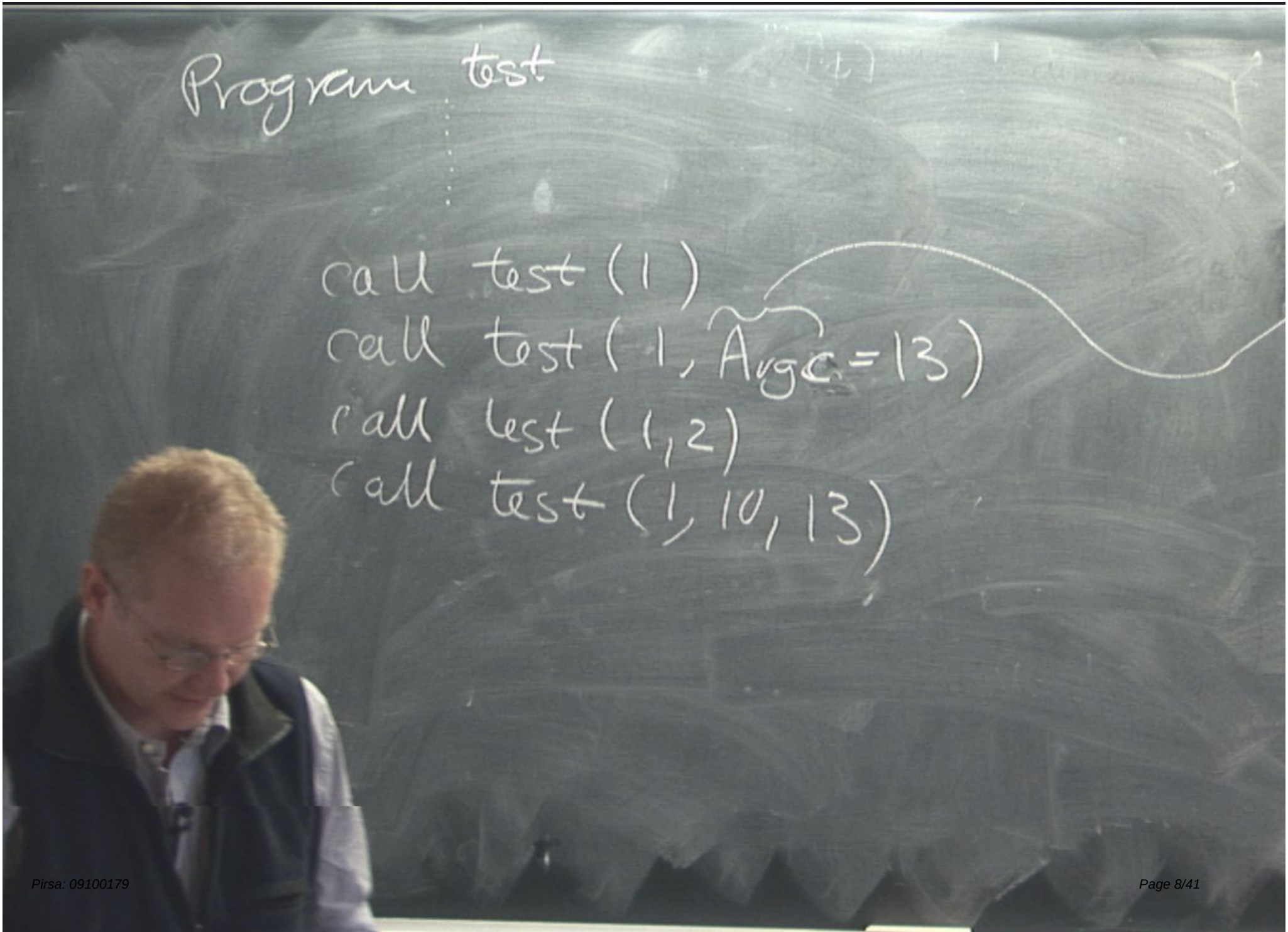
Optional Arguments

If we declare an explicit face

```
Interface
    subroutine test(Arga, Argb, Argc)
        integer, intent(inout) :: Arga
        integer, intent(inout), optional :: Argb, Arg
    end subroutine test
end interface
```

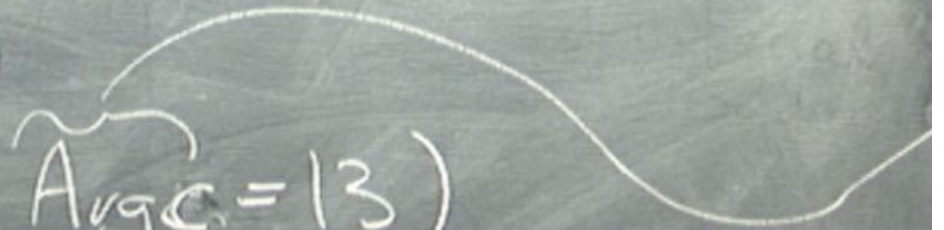

Program test
call test (1)

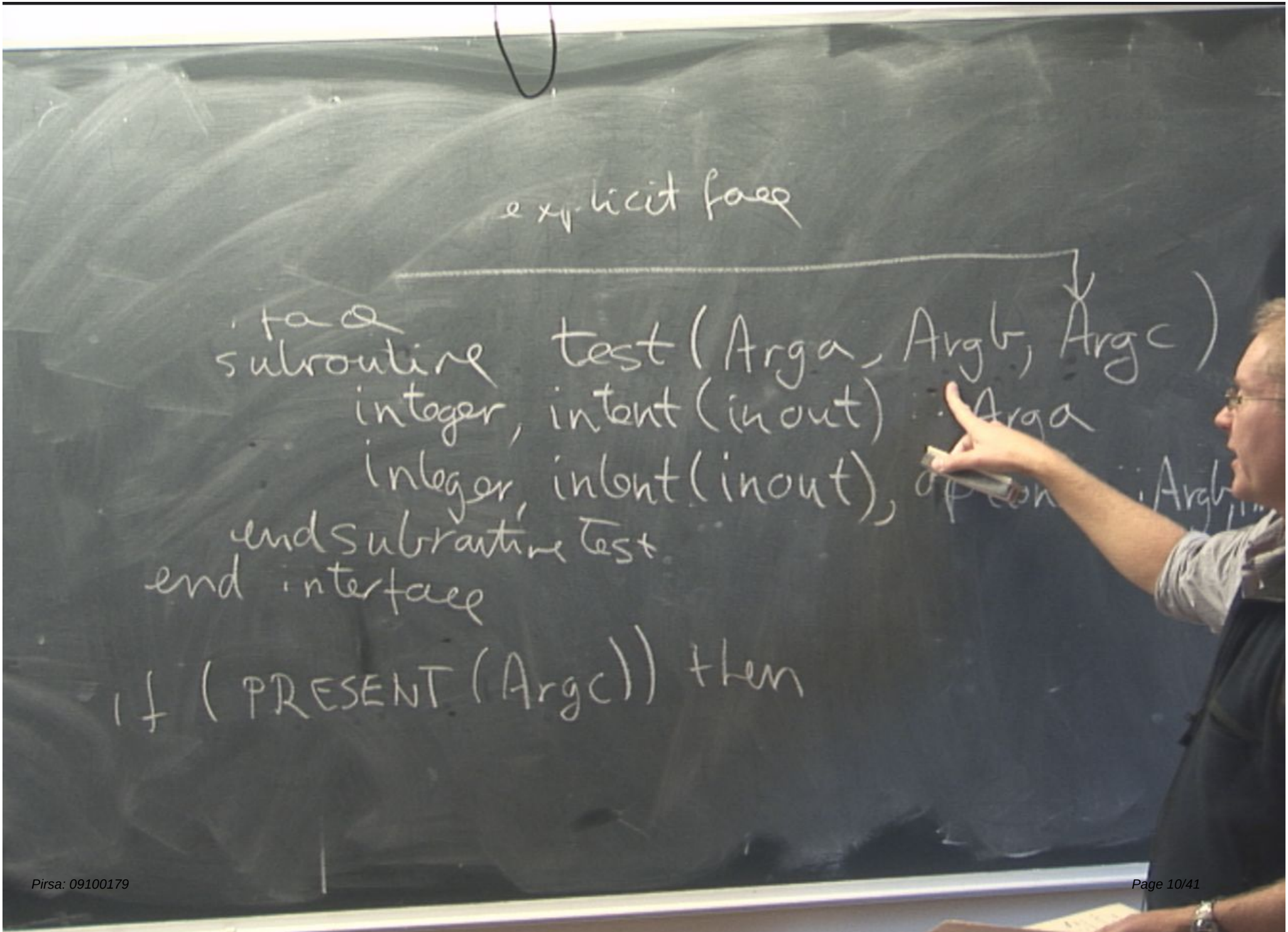

Program test
call test(1)
call test(1, Avg=13)

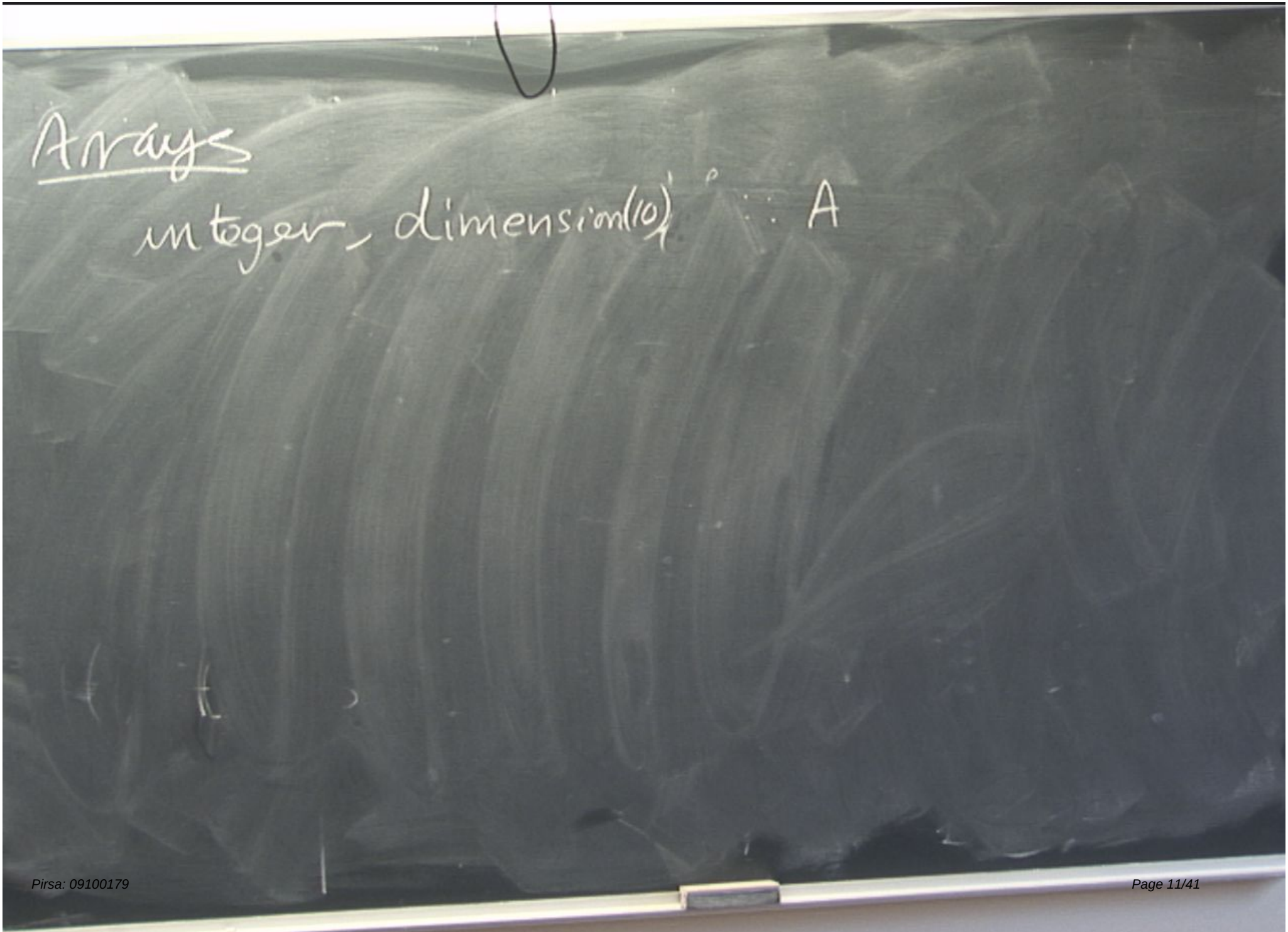

Program test
call test(1)
call test(1, Avg c. = 13)
call test(1,2)
call test(1, 10, 13)

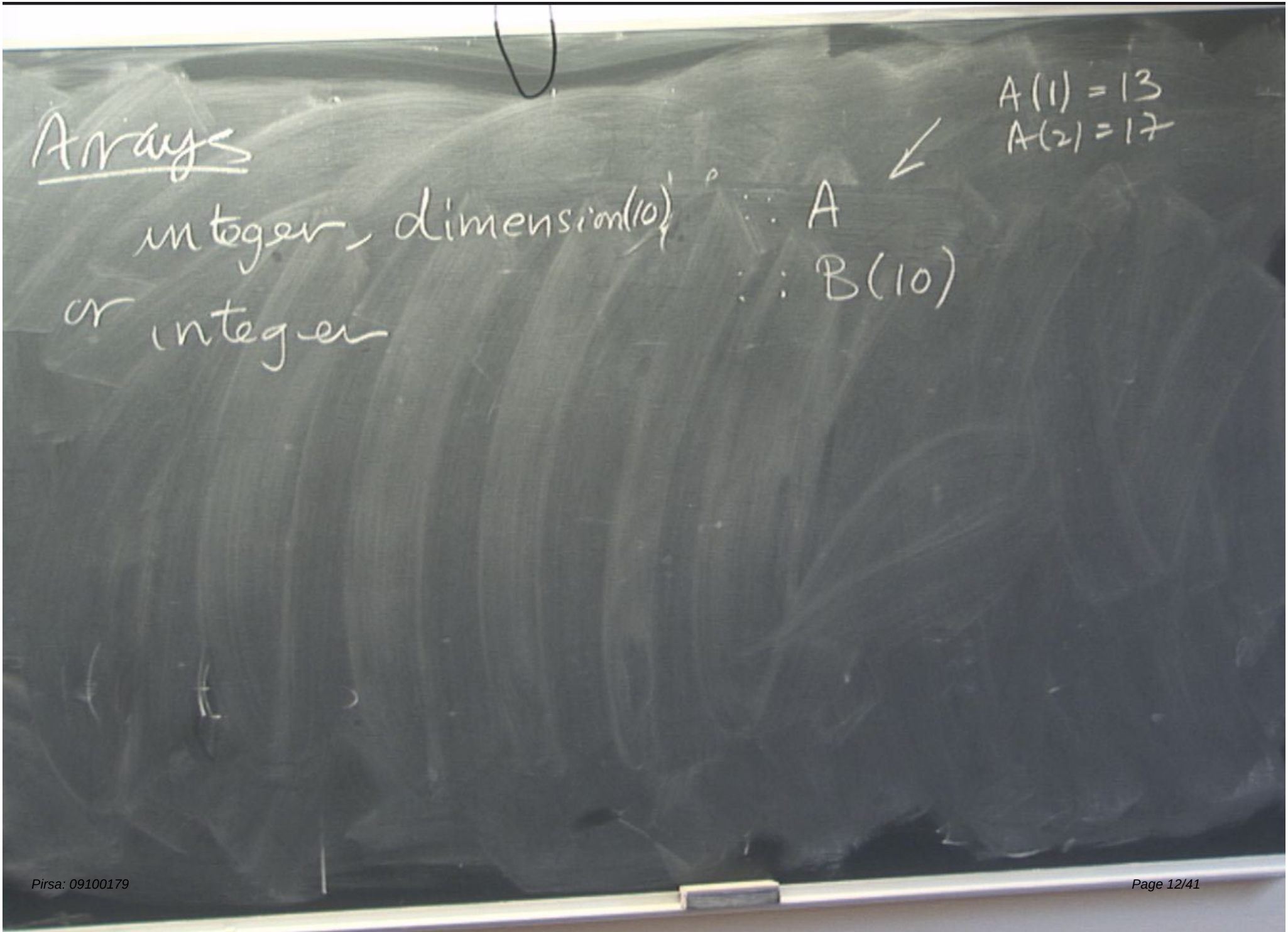
Program test

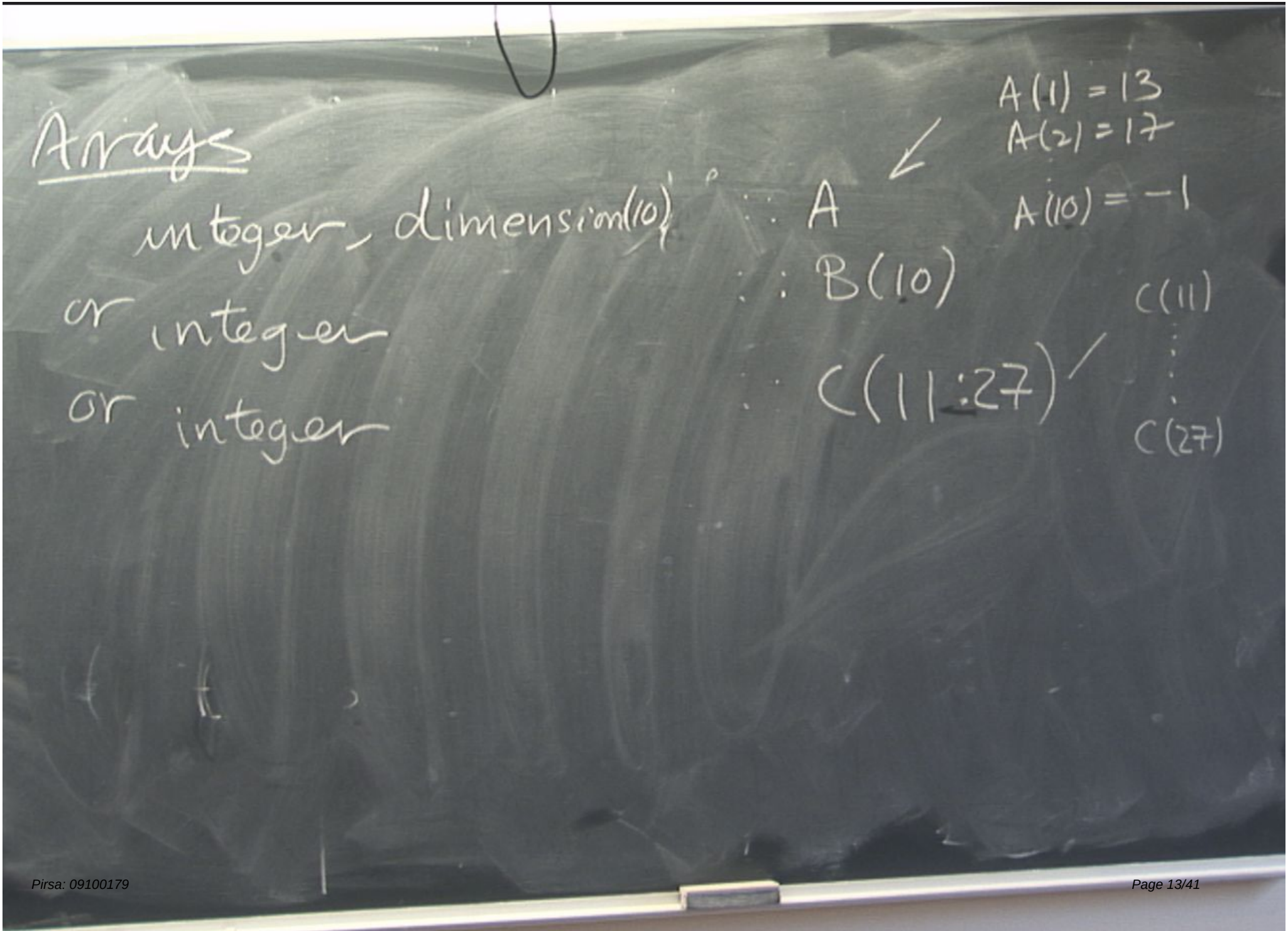
call test(1)
call test(1, Argc=13)
call test(1,2)
call test(1,10,13)
call test(1, Argc=13, Arb=10)
(Argc=13, Arga=1)




explicit face
interface
subroutine test(Arga, Argb, Argc)
integer, intent(inout) :: Arga
integer, intent(inout), optional :: Argb
end subroutine test
end interface
if (PRESENT(Argc)) then


Arrays
integer, dimension(10) :: A


Arrays
integer, dimension(10) :: A
or integer :: B(10)
A(1) = 13
A(2) = 17


Arrays
integer, dimension(10) :: A
or integer :: B(10)
or integer :: C(11:27)
A(1) = 13
A(2) = 17
A(10) = -1
C(11)
C(27)

Arrays

integer, dimension(10) :: A

or integer :: B(10)

or integer :: C(11:27)

or integer :: D(-2:45)

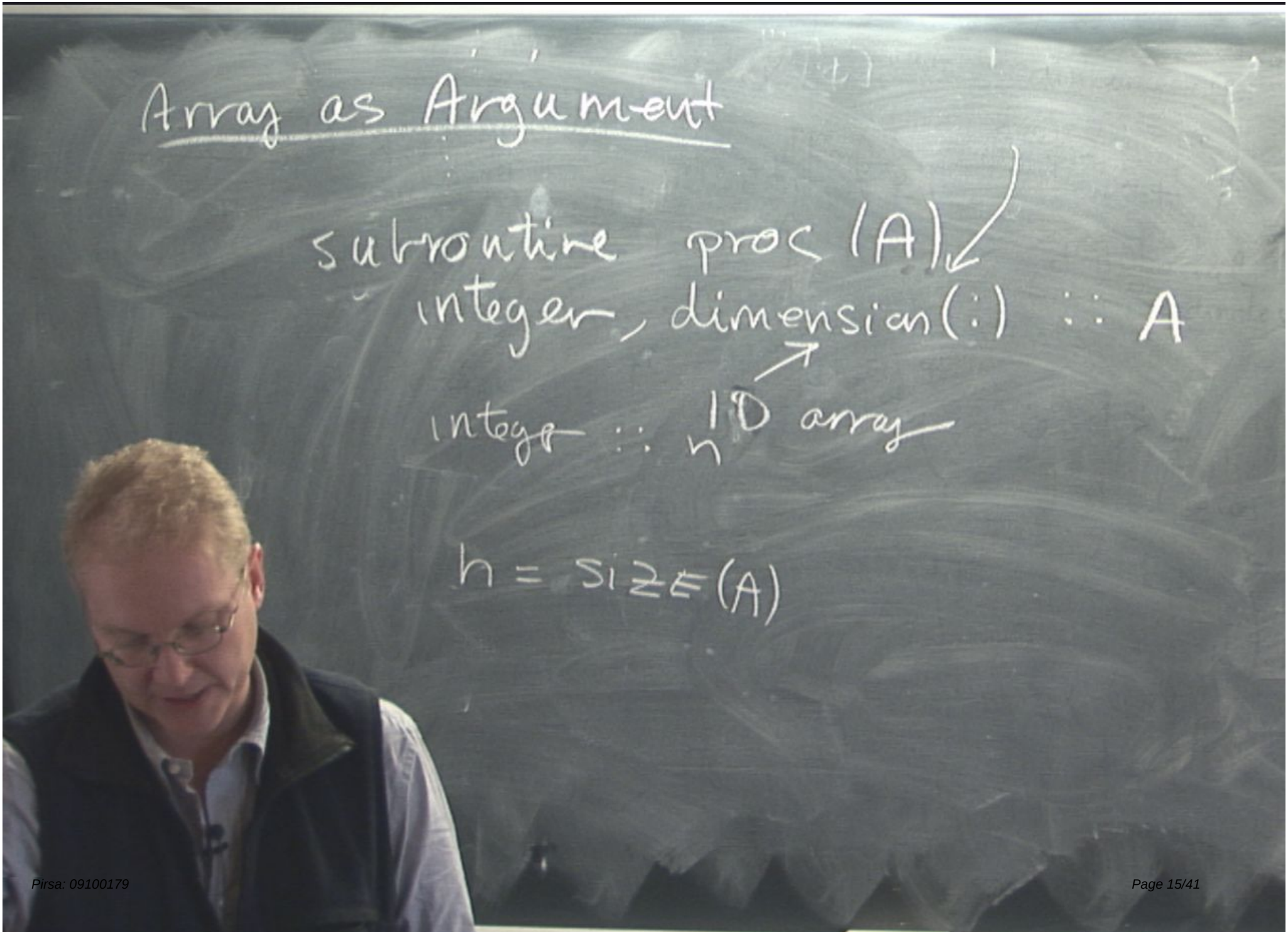
A(1) = 13

A(2) = 17

A(10) = -1

C(11)

C(27)


Array as Argument
subroutine proc (A)
integer, dimension(:) :: A
integer :: n
1D array
n = size(A)

Array as Argument

Assumed-sh

```
subroutine proc (A)
  integer, dimension(:) :: A
```

integer :: n 1D array

```
n = size(A)
```

Array as Argument

Assumed-shape

```
subroutine proc (A)
  integer, dimension(:) :: A
  integer :: n
  n = size(A)
```

1D array

reference to the Array

Array as Argument

Assumed-shape

```
subroutine proc(A)
  integer, dimension(:) :: A
  integer :: n
```

1D array

```
n = size(A)
n = Lbound(A,1)
n = Ubound(A,1)
```

first dimension

Storing in Memory

integer :: A(3,3) 3x3 matrix

is stored column major

A(1,1), A(2,1), A(3,1), A(1,2), A(2,2), A(3,2)

C uses row_major

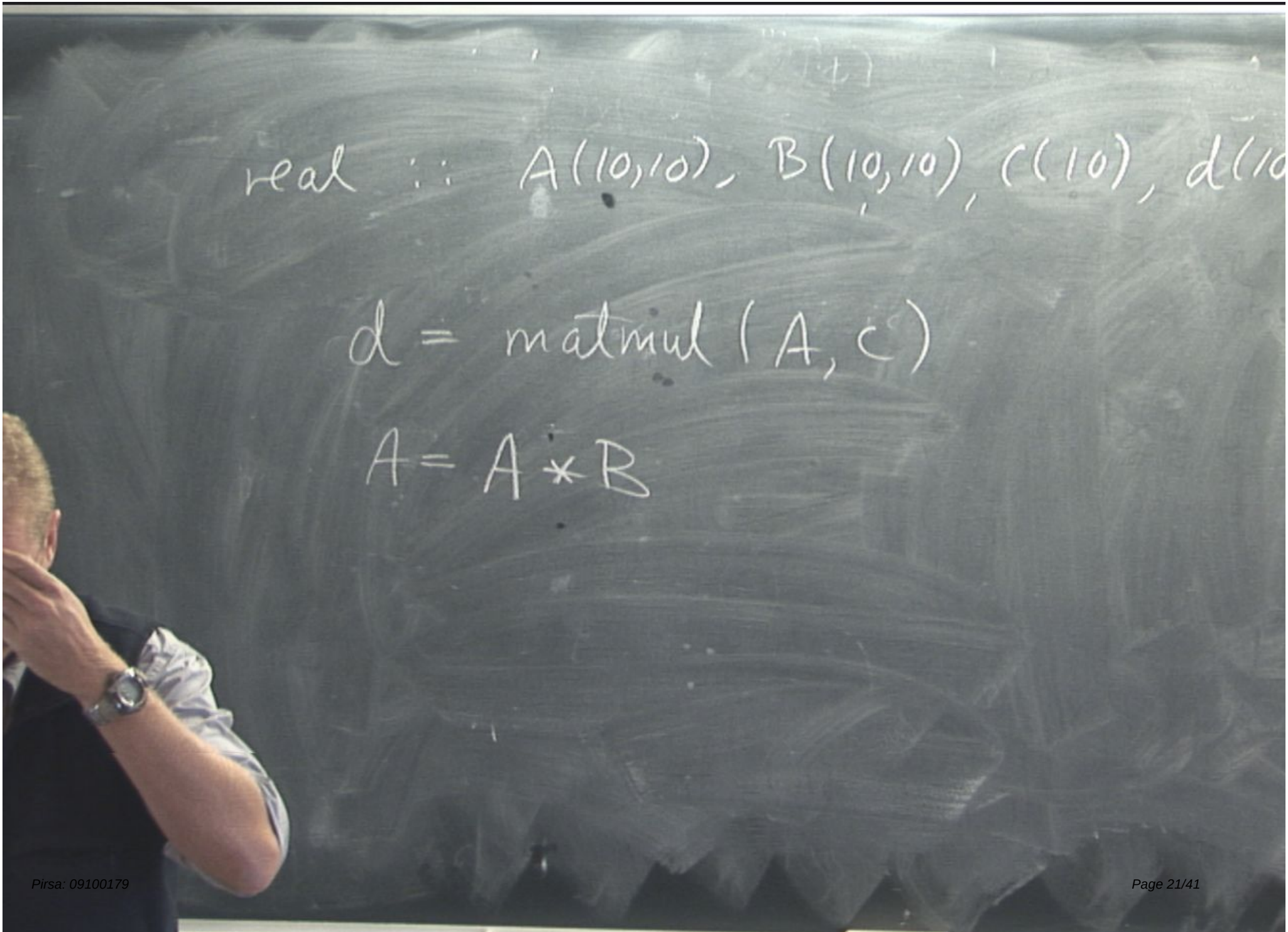
Storing in Memory

integer :: A(3,3) 3x3 matrix

is stored column major

A(1,1), A(2,1), A(3,1), A(1,2), A(2,2), A(3,2)

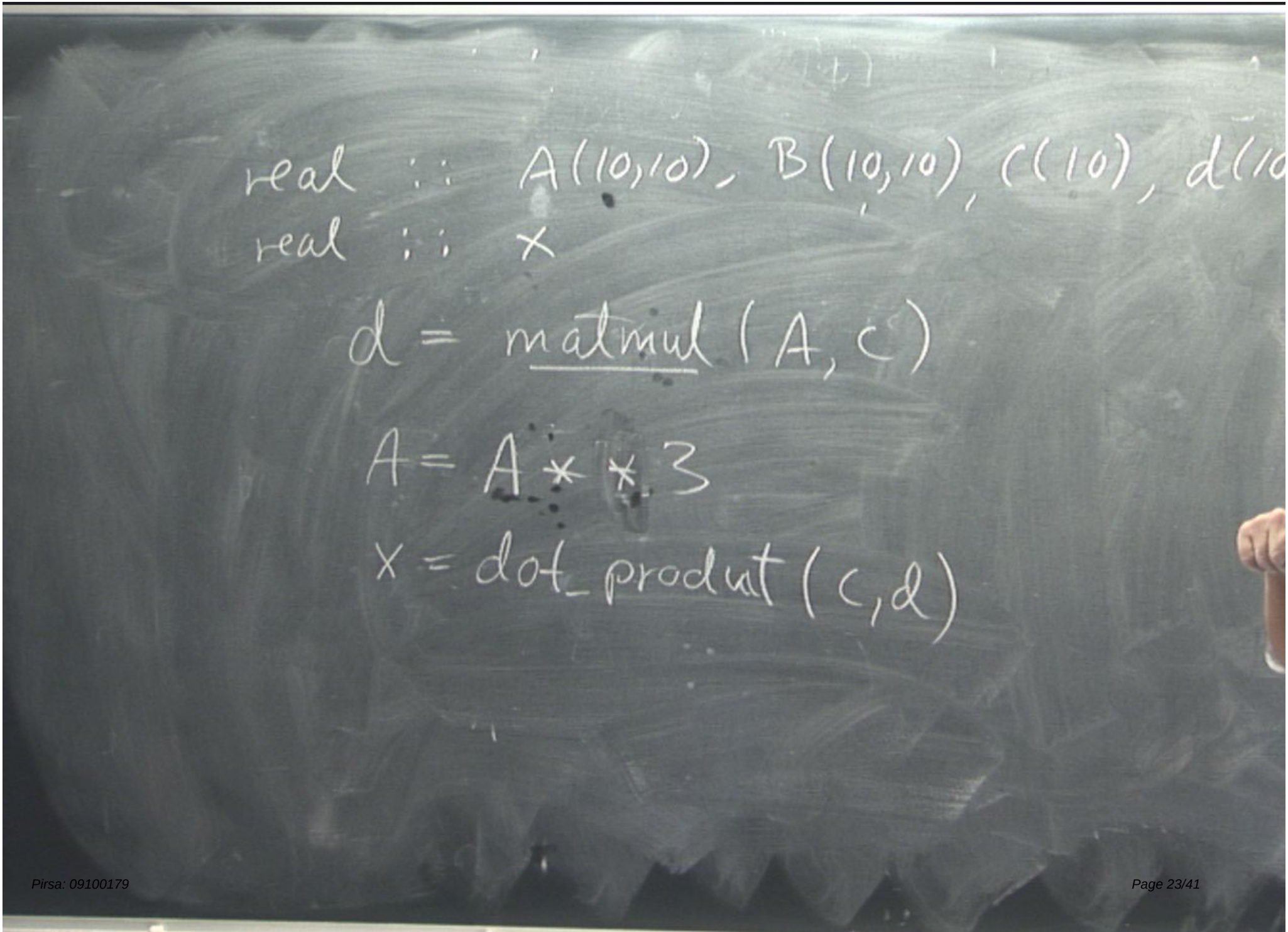
C uses row_major


real :: A(10,10), B(10,10), C(10), d(10
d = matmul(A, C)
A = A * B

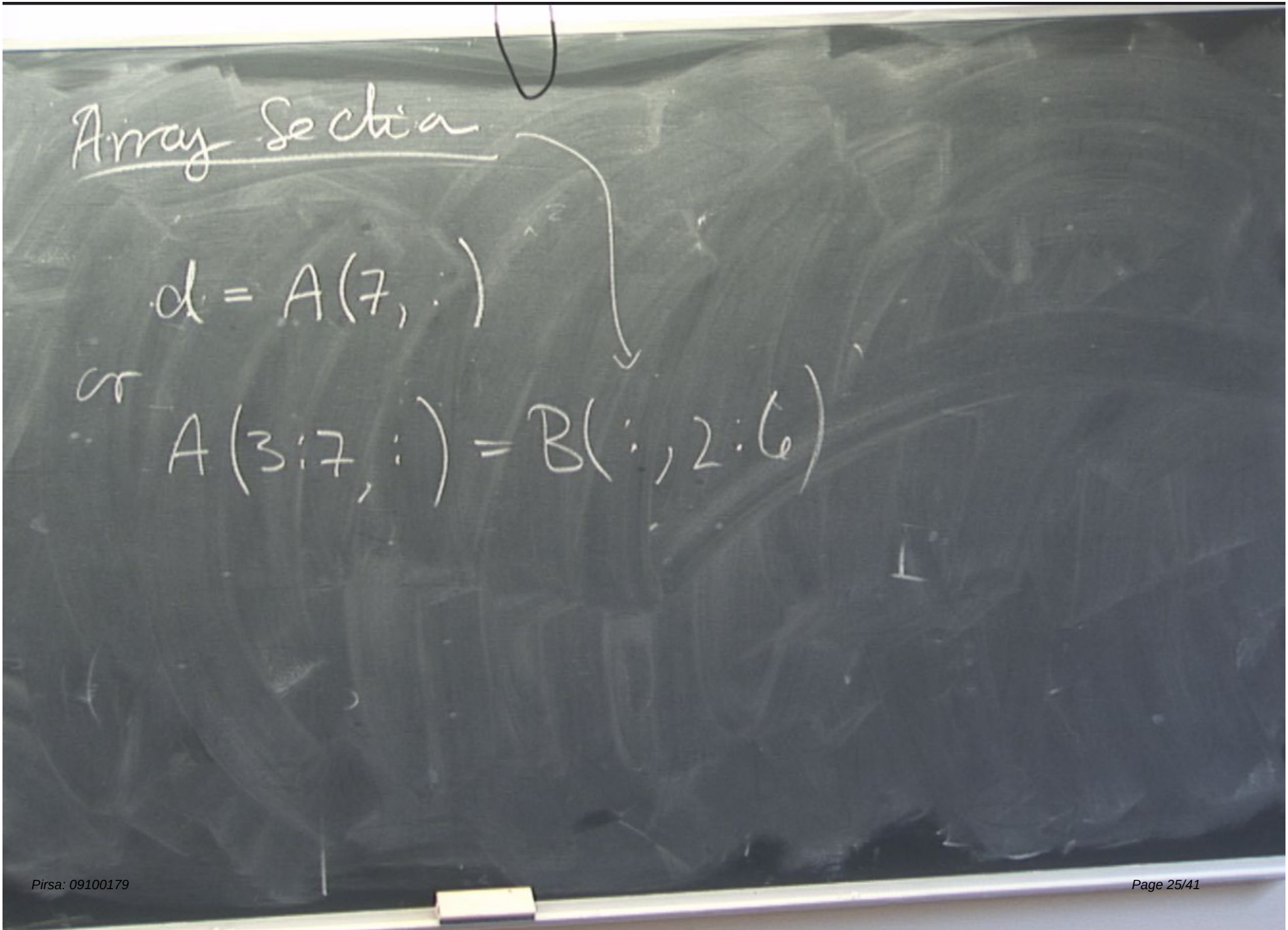
```
real :: A(10,10), B(10,10), C(10), d(10
```

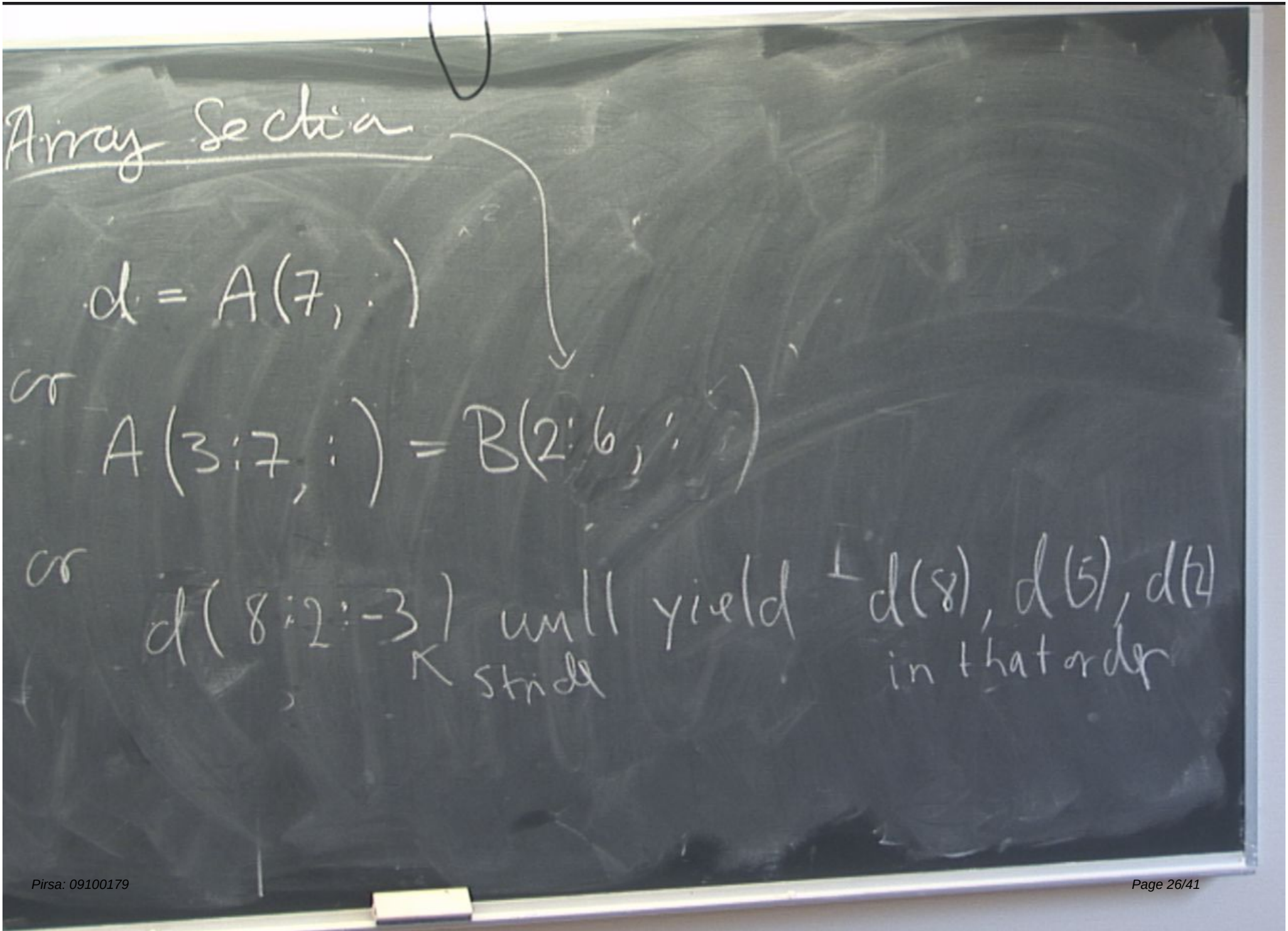
```
d = matmul(A, C)
```

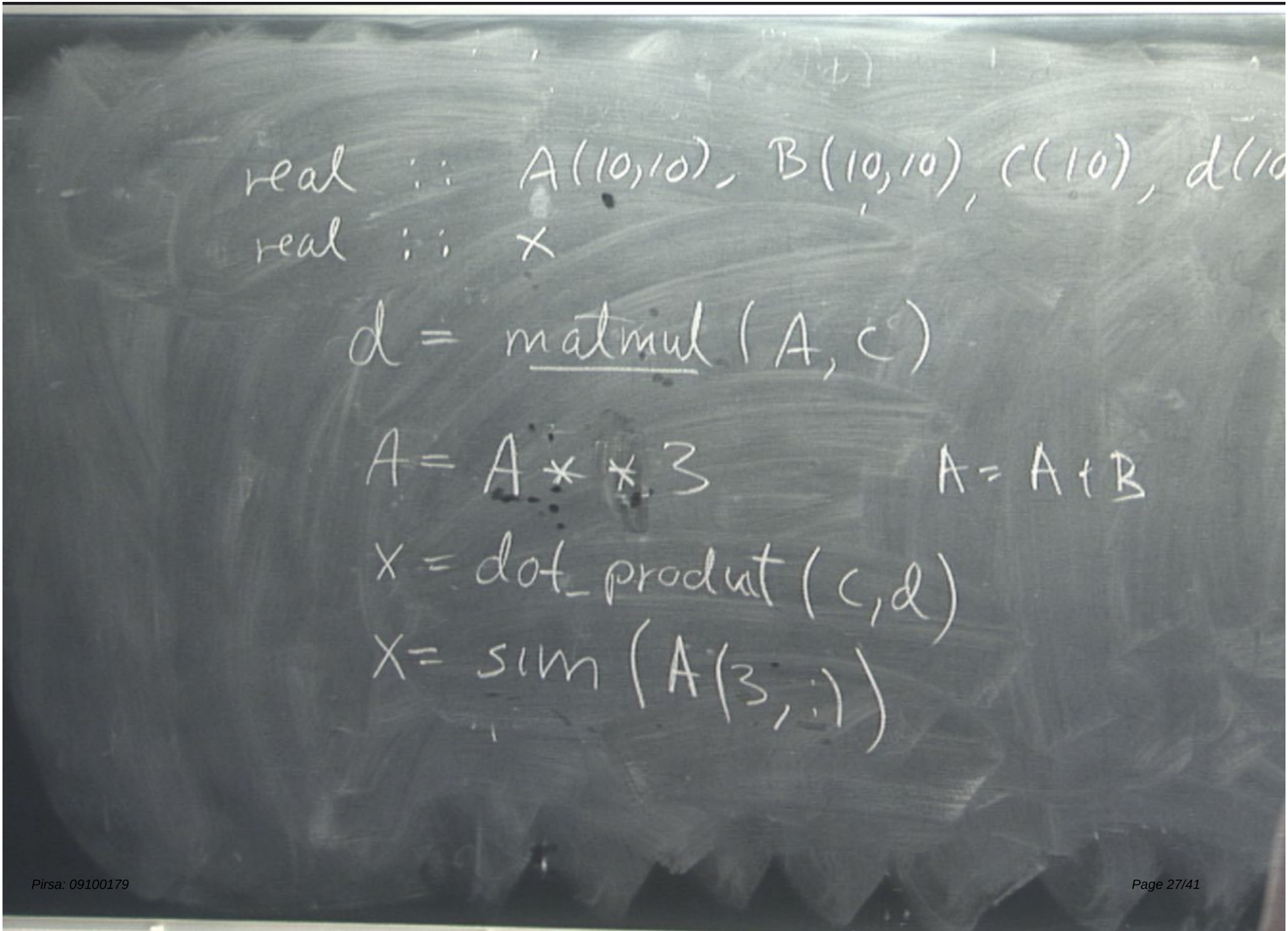
```
A = A**3
```

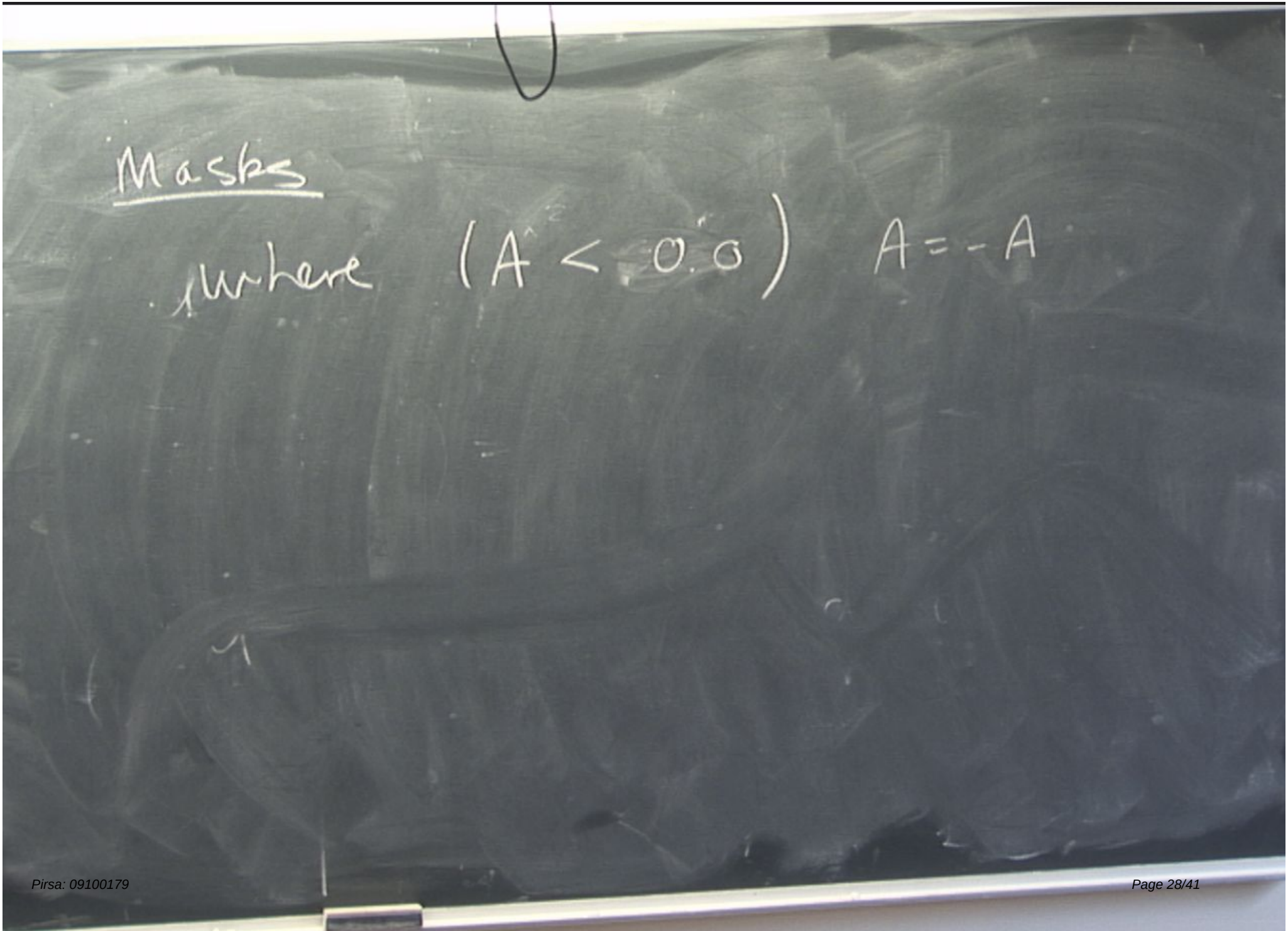

real :: A(10,10), B(10,10), C(10), d(10
real :: x
d = matmul(A, C)
A = A ** 3
x = dot_product(C, d)

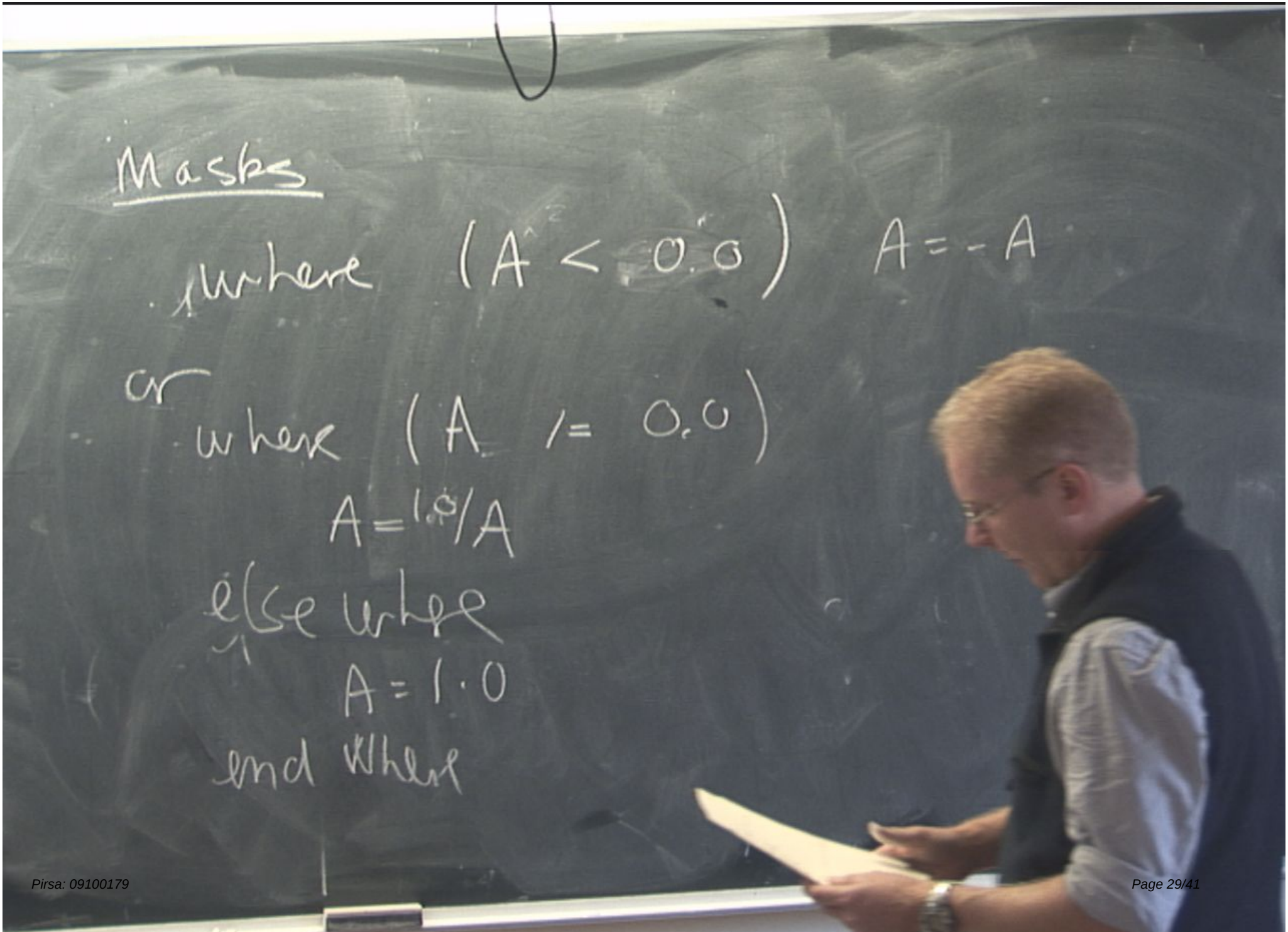
```
real :: A(10,10), B(10,10), C(10), d(10
real :: x

d = matmul(A, c)

A = A**3

x = dot_product(c,d)

x = sum(A(3,:))
```


Array Section
d = A(7, :)
or
A(3:7, :) = B(:, 2:6)


Array Section
d = A(7, :)
or
A(3:7, :) = B(2:6, :)
or
d(8:2:-3) will yield d(8), d(5), d(2)
stride
in that order

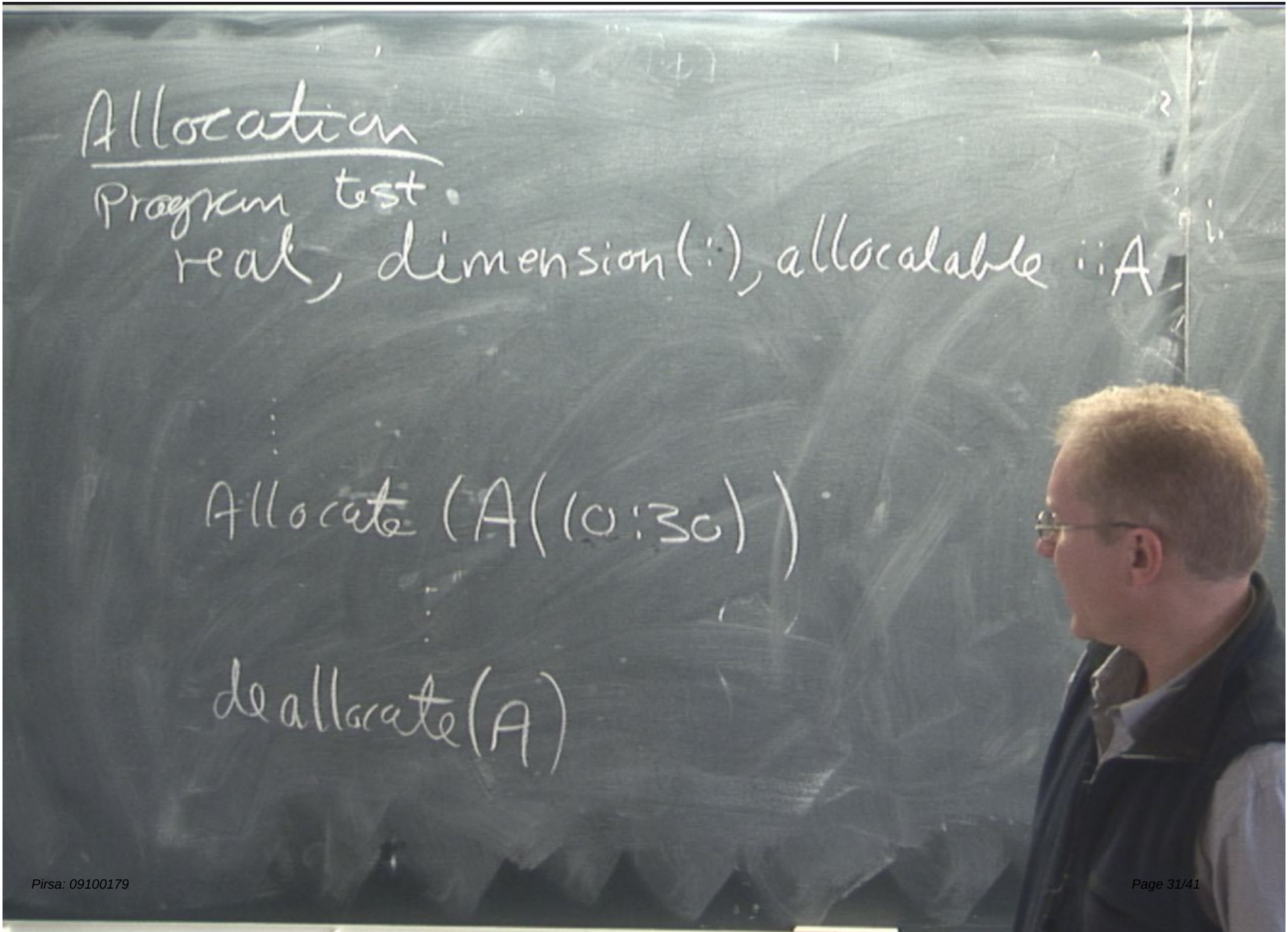

real :: A(10,10), B(10,10), C(10), d(10
real :: x
d = matmul(A, c)
A = A ** 3
A = A + B
x = dot_product(c, d)
x = sum(A(3, :))


Masks
where (A < 0.0) A = -A


Masks
where (A < 0.0) A = -A
or
where (A /= 0.0)
A = 1.0/A
elsewhere
A = 1.0
end where

Allocation

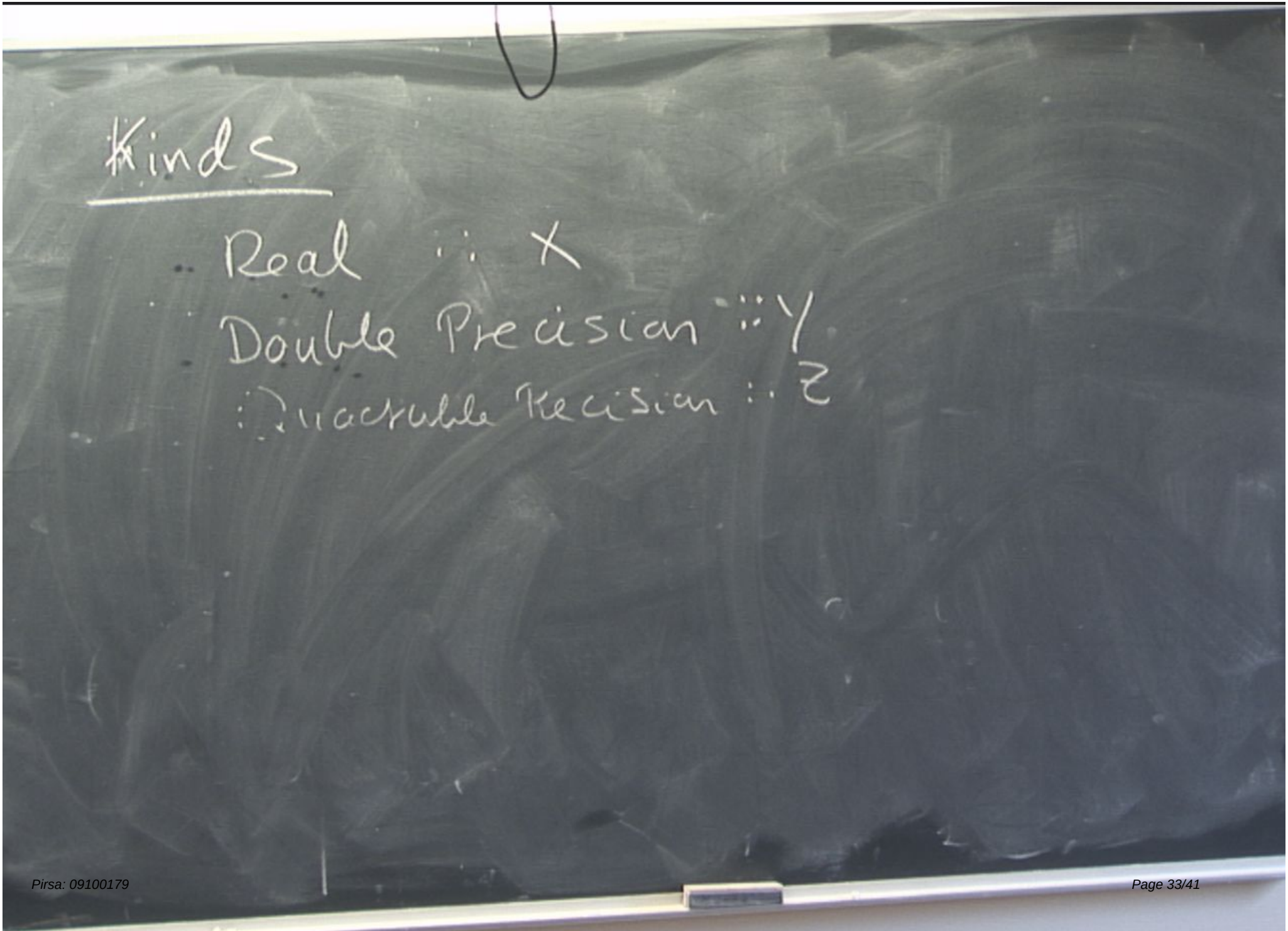
```
Program test
  real, dimension(:), allocatable :: A
```

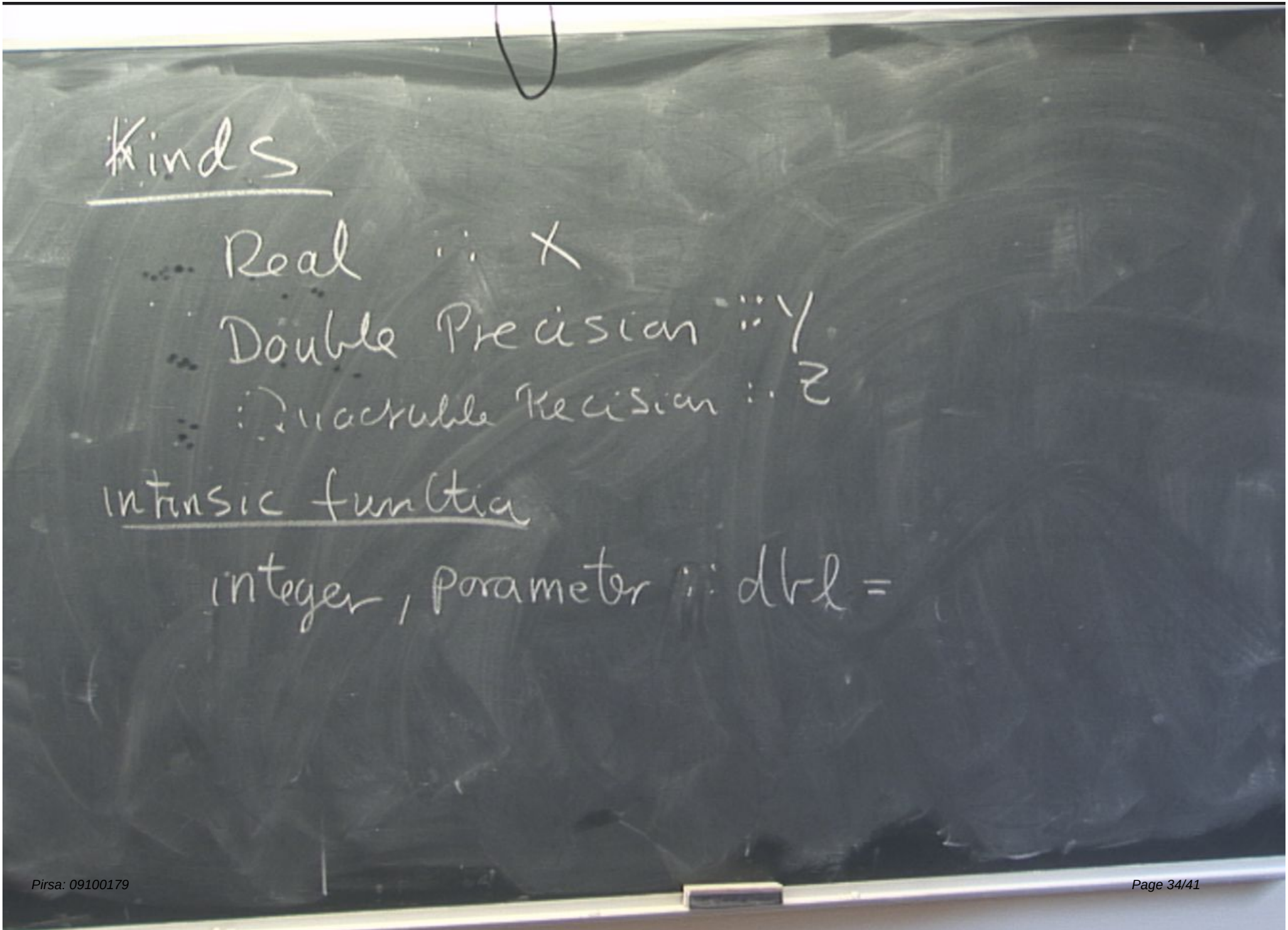

Allocation
Program test
real, dimension(:), allocatable :: A
Allocate (A((0:30))
deallocate(A)

Allocation

```
Program test
  real, dimension(:), allocatable :: A
  ⋮
  Allocate (A((0:30)))
  ⋮
  deallocate(A)
```

Always combine


Kinds
Real :: X
Double Precision :: Y
Quadruple Precision :: Z


Kinds
Real :: x
Double Precision :: y
Quadruple Precision :: z
intrinsic functions
integer, parameter :: dbl =

Kinds

Real :: x

Double Precision :: y

Quadruple Precision :: z

Intrinsic functions

integer, parameter :: dbl = selected_real_kind(p,r)

Kinds

Real :: x

Double Precision :: y

Quadruple Precision :: z

intrinsic function

integer, parameter :: dbl = selected_real_kind(p, r)

p: minimal number of decimal dig

r: minimal range of exponent

Kinds

Real :: x

Double Precision :: y

Quadruple Precision :: z

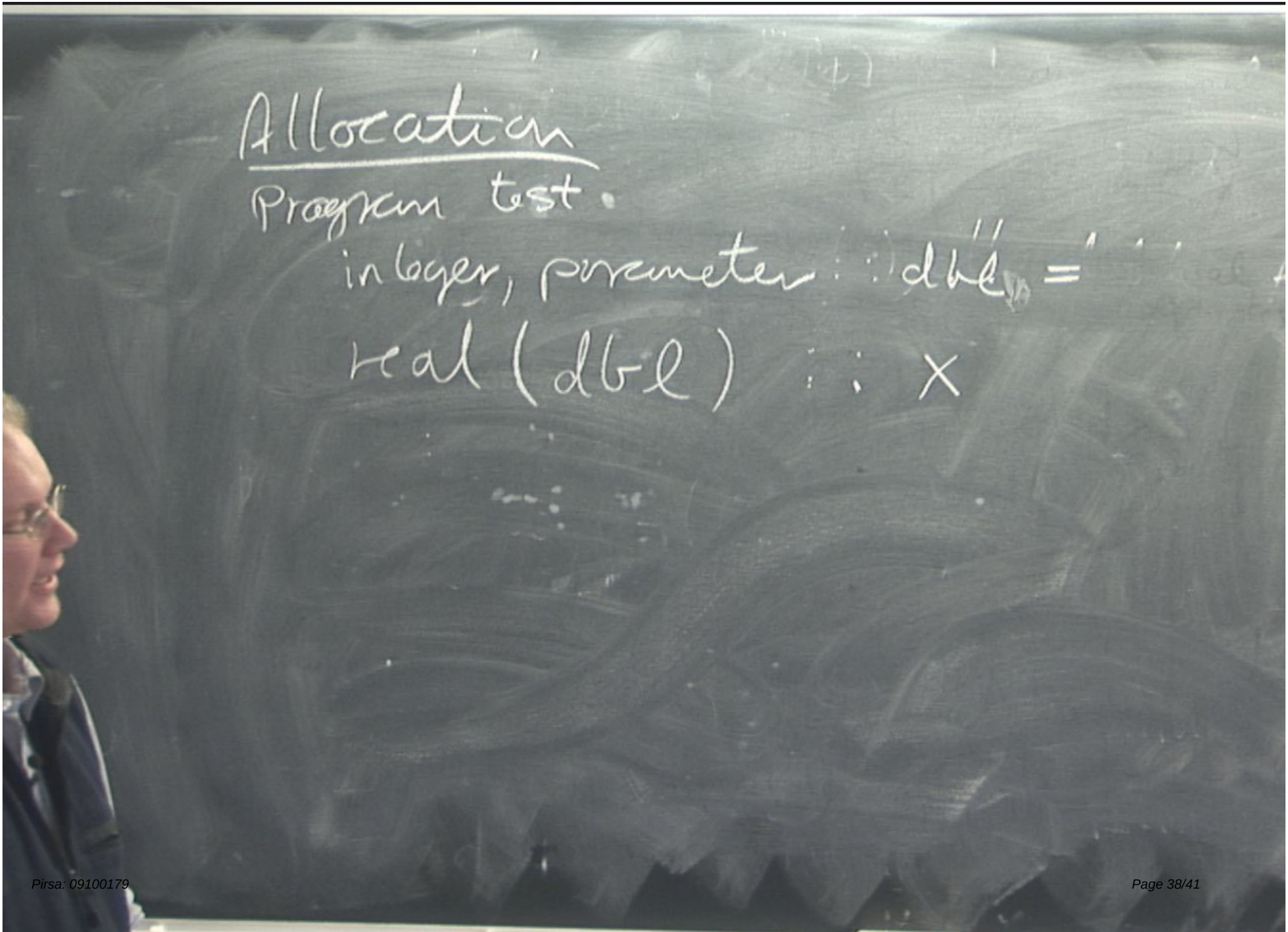
Intrinsic function

integer, parameter :: dbl = selected_real_kind(p, r)

kind that correspond

↳ minimal number of decimal dig

↳ r; minimal range of exponent


Allocation
Program test.
integer, parameter :: dbl =
real (dbl) :: x

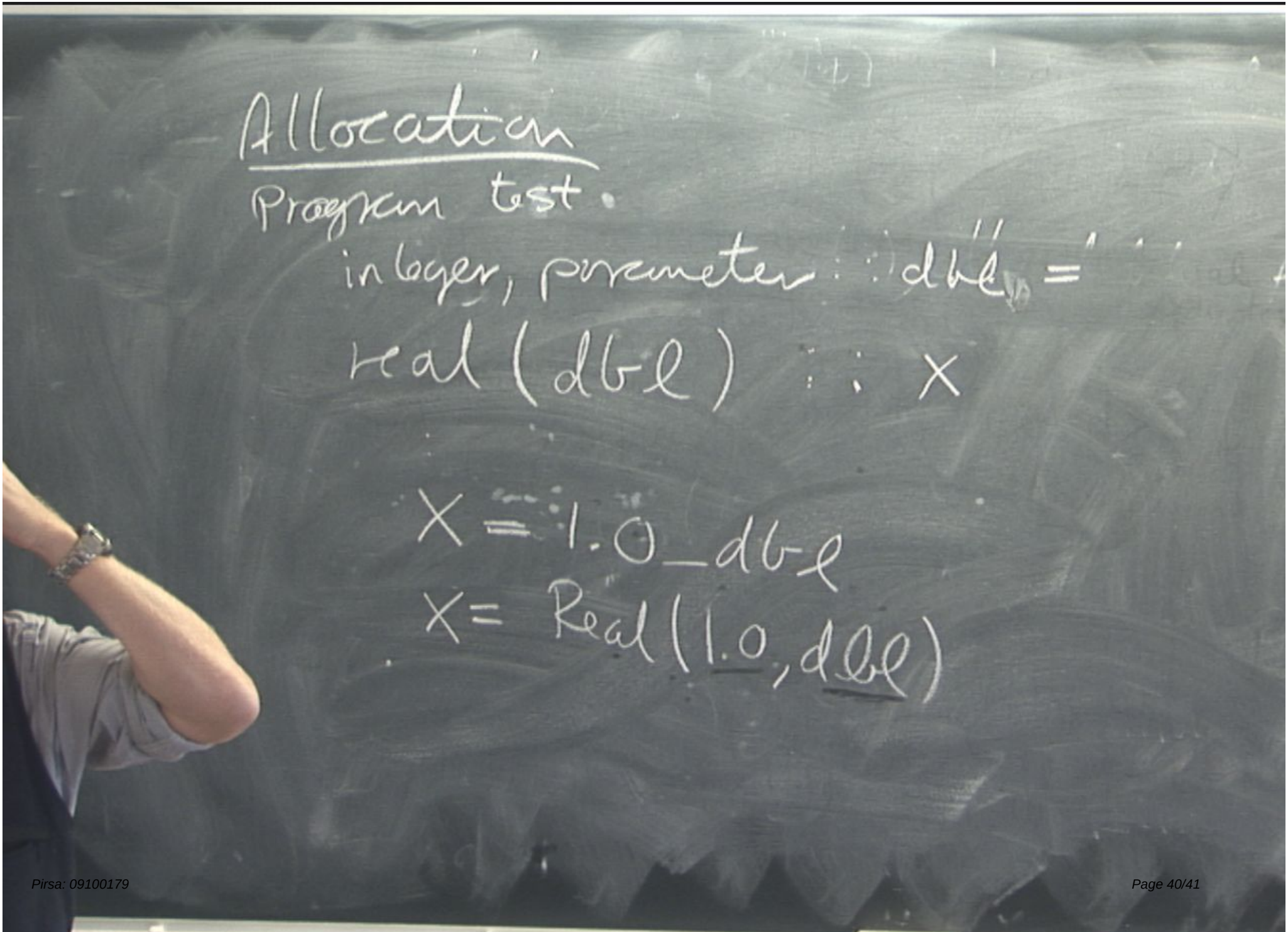
Allocation

Program test.

integer, parameter :: dbl = "

real (dbl) :: x

x = 1.0


Allocation
Program test.
integer, parameter :: dbl =
real (dbl) :: x
x = 1.0_dbl
x = Real (1.0, dbl)

Kinds

selected_int_kind(r)

$-10^r < i < 10^r$

Real :: x

Double Precision :: y

Quadruple Precision :: z

kind that corresponds

ntrinsic functia

integer, parameter :: dbl = selected_real_kind(p, r)

p: minimal number of decimal digits

r: minimal range of exponent