i116: Basic of Programming

2. Some program constructs (2)

Kazuhiro Ogata, Canh Minh Do

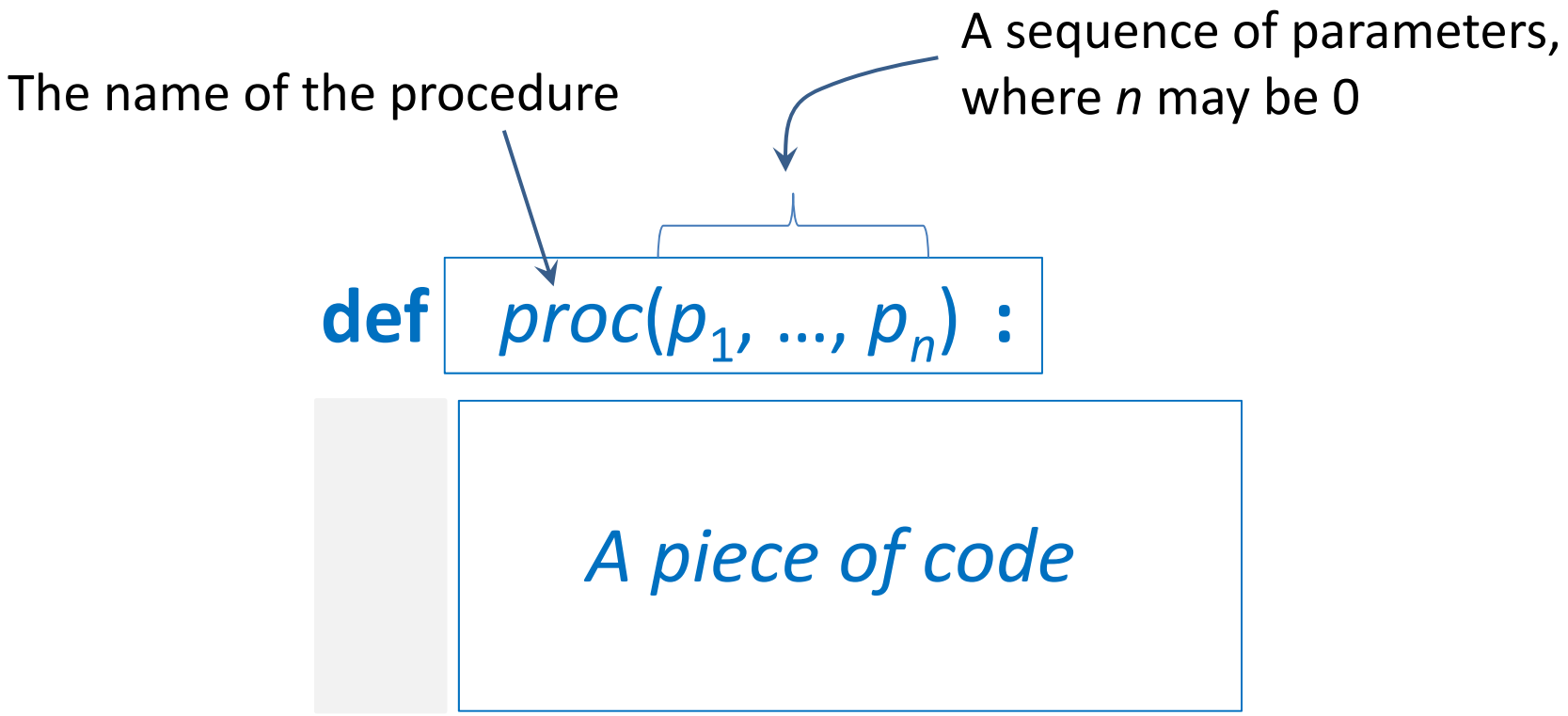
Roadmap

- Procedures (or functions)
- Module (or file) importation
- Exception handling
- Some more examples

Procedures (or functions)

- In addition to $5!$, we would like to calculate $n!$ for some n.
- We may want to calculate $5!$ more than once in a program.
- We can name a piece of code as a *procedure*.
- A procedure may have parameters.

Procedures (or functions)



Procedures (or functions)

```
def factLoop(n):
    redBox = 1
    blueBox = 1
    while blueBox < n or blueBox == n:
        redBox = redBox * blueBox
        blueBox = blueBox + 1
    return redBox
```

Procedures (or functions)

```
print('5! = ', factLoop(5))
print('0! = ', factLoop(0))
print('10! = ', factLoop(10))
print('100! = ', factLoop(100))
```

Recursive calls

- A procedure can call itself in its piece of code.
- *Recursive procedures* can be defined, or *recursive calls* are doable.

def *proc*(p_1, …, p_n) **:**

    … *proc*(…) …

Recursive calls

```
def factRecur(n):
    if n == 0:
        return 1
    else:
        return n * factRecur(n-1)
```

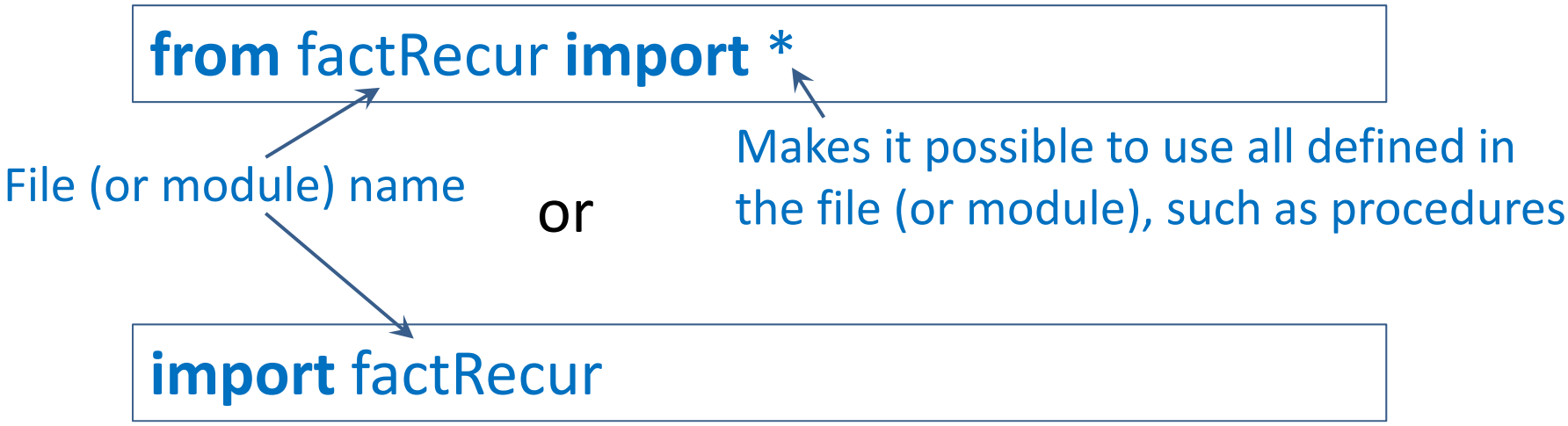
$$n! = \begin{cases} 1 & \text{if } n = 0 \\ n * (n\text{-}1)! & \text{otherwise} \end{cases}$$

Recursive calls

```
print('5! = ', factRecur(5))
print('0! = ', factRecur(0))
print('10! = ', factRecur(10))
print('100! = ', factRecur(100))
```

Module importation

- Let us suppose that factRecur is written in a file whose name is factRecur.py.
- To use factRecur in another program written in another file, we should write the following in the other file before use of factRecur:



Module importation

```
from factRecur import *

print('n! is calculated.')
s = input('Please input n: ')
n = int(s)
print(n, '! = ', factRecur(n))
```

A user is asked to input something, which is treated as a string.

A string is converted into an integer, provided that the string only consists of numbers.

Exception handling

- What if a user inputs -1?
- factRecur(-1) is calculated, which never terminates.
- In real, the maximum recursion depth exceeds and then Python complains it.
- So, we will first revise factRecur(…) to avoid the exceeding of the maximum recursion depth.

Exception handling

An exception is raised and the warning or error message is displayed.

```
def revFactRecur(n):
    if n < 0:
        raise Exception('The argument must be a natural number, such as 0, 1 and 2!')
    elif n == 0:
        return 1
    else:
        return n * revFactRecur(n-1)
```

Exception handling

```
try:
    block1 (or fragment1)
except Exception :
    block2 (or fragment2)
```

Whenever an exception is raised in *block1*, the control goes to *block2*, meaning *block2* is done.

Exception handling

```
from revFactRecur import *

print('n! is calculated.')
while True:
    try:
        s = input('Please input n: ')
        n = int(s)
        print(n, '! = ', revFactRecur(n))
        break
    except Exception:
        pass
```

If we write nothing in a block, a fragment or a piece of code, we should write **pass**.

Exception handling

try:

    *block*1 *(or fragment*1*)*

except Exception **as** *em*:

    block2 (or fragment2)

em refers to the warning or error message when an exception is raised in *block*1 (or *fragment*1) and can be used in *block2* (or *fragment2*)

Exception handling

```
from revFactRecur import *

print('n! is calculated.')
while True:
    try:
        s = input('Please input n: ')
        n = int(s)
        print(n, '! = ', revFactRecur(n))
        break
    except Exception as em:
        print(em)
```

Exception handling

- What if a user inputs -1?
- The following message is displayed and a user is prompted to input another:

```
Please input n: -1
The argument must be a natural number, such as 0, 1 and 2!
Please input n:
```

Exception handling

- What if a user inputs abc?
- Another exception is raised by int(…) and handled by the program.
- The following message is displayed and a user is prompted to input another:

```
Please input n: abc
invalid literal for int() with base 10: 'abc'
Please input n:
```

Exception handling

- What if a user inputs 1000?
- The maximum recursion depth exceeds, yet another exception is raised by Python and handled by the program.
- The following message is displayed and a user is prompted to input another:

```
Please input n: 1000
maximum recursion depth exceeded
Please input n:
```

Some more examples

```
def srByLinearSearch(v0):
    for i in range(v0):
        if i * i > v0:
            return i - 1
```

```
print('The square root of ', 200000000, ' is ',
srByLinearSearch(200000000), '.')
print('The square root of ',
20000000000000000, ' is ',
srByLinearSearch(20000000000000000), '.')
```

Some more examples

```
def srByBinarySearch(v0):
    v1 = 0
    v2 = v0
    while v1 != v2:
        if (v2-v1)%2 == 0:
            v3 = v1+(v2-v1)//2
        else:
            v3 = v1+(v2-v1)//2+1
        if v3*v3 > v0:
            v2 = v3-1
        else:
            v1 = v3
    return v1
```

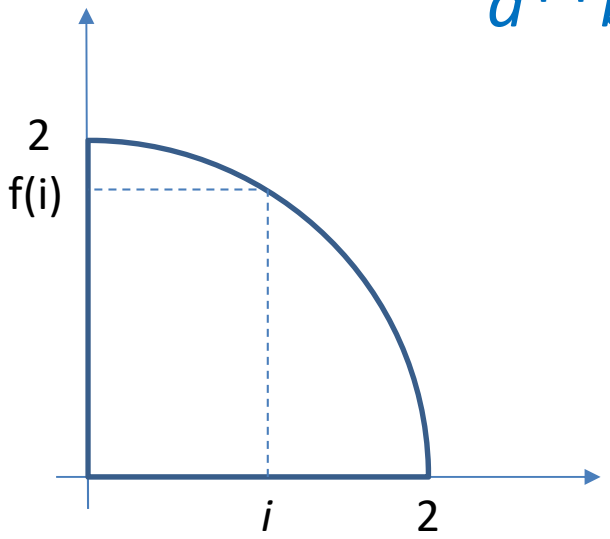
Some more examples

```
print('The square root of ', 200000000, ' is ',
srByBinarySearch(200000000), '.')
print('The square root of ',
20000000000000000, ' is ',
srByBinarySearch(20000000000000000), '.')
```

Some more examples

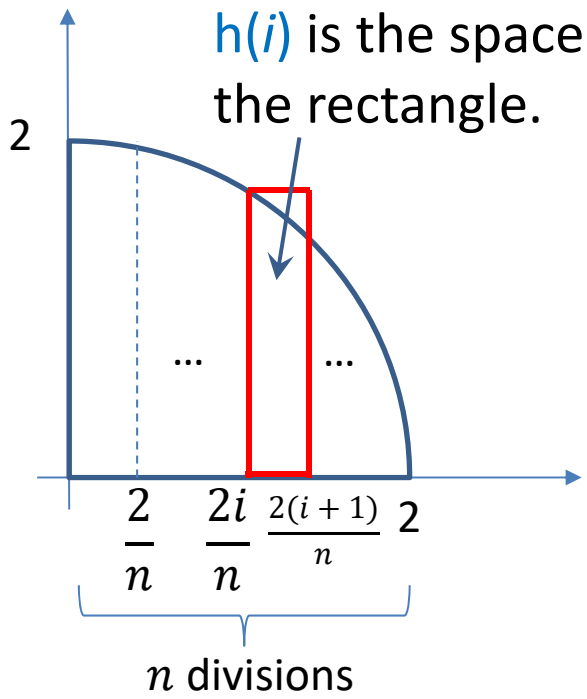
```
def f(x):
    return (4 - x**2)**(1/2)
```

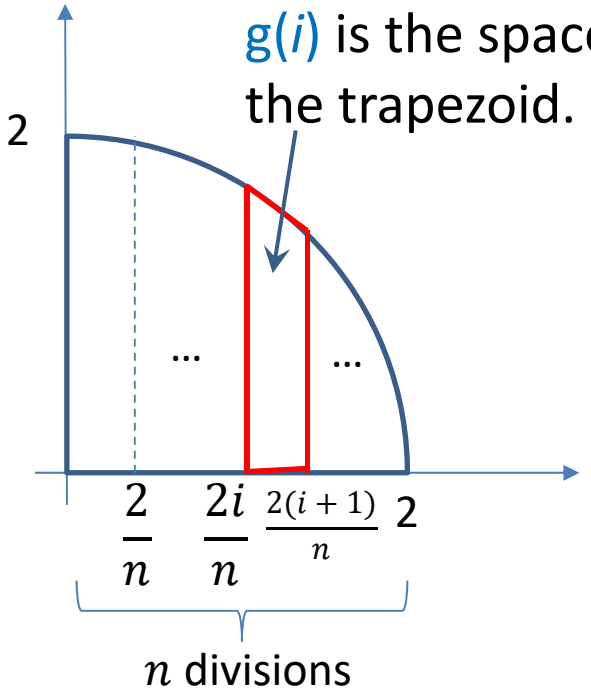
$a**b$ means a^b.



Some more examples

```
def h(i,n):
    return f(i*(2/n)) * (2/n)
def g(i,n):
    return (f(i*(2/n)) + f((i+1)*(2/n))) * (2/n) * (1/2)
```





Some more examples

```
def piWithRectangle(e):
    v1 = 0
    v2 = f(0) * 2
    n = 1
    while abs(v1 - v2) > e:
        n = 2 * n
        v1 = v2
        v2 = 0
        for i in range(n):
            v2 = v2 + h(i,n)
    return v2
```

Some more examples

```
print('pi is ', piWithRectangle(0.001), ', where
e is 0.001.')
print('pi is ', piWithRectangle(0.000001), ',
where e is 0.000001.')
print('pi is ', piWithRectangle(0.00000001), ',
where e is 0.00000001.')
```

Some more examples

```
def piWithTrapezoido(e):
    v1 = 0
    v2 = (f(0) + f(1)) * 2 * (1/2)
    n = 1
    while abs(v1 - v2) > e:
        n = 2 * n
        v1 = v2
        v2 = 0
        for i in range(n):
            v2 = v2 + g(i,n)
    return v2
```

Some more examples

```
print('pi is ', piWithTrapezoido(0.001), ',
where e is 0.001.')
print('pi is ', piWithTrapezoido(0.000001), ',
where e is 0.000001.')
print('pi is ', piWithTrapezoido(0.00000001), ',
where e is 0.00000001.')
```