CSE 1061 **Introduction to Computing**

Lecture 7

Fall 2015

Department of Computing
The School of EE & Computing
Adama Science & Technology University

OUTLINE

Functions

Built-in functions and modules

User-defined functions

Case study: Refining the pseudo coding !!: decomposition and abstraction

Keyboard input

Reading assignment

Chapter 4 of the textbook

Tutorial on cs1graphics(Chapter 3)

http://www.cs1graphics.org/

FUNCTIONS

ASTU

- The name **function** comes from mathematics. A **function** is a **mapping** from one set(**domain**) to another set(**range**):

f: X→ Y such that y = f(x),
where x ∈ X and y ∈ Y.

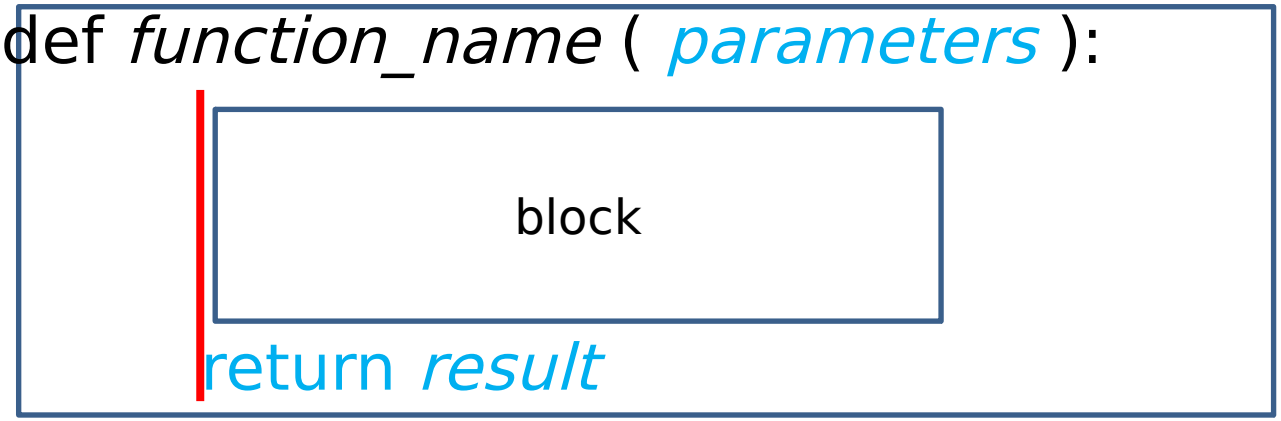
x is the **argument** of the function., and *f* (*x*) is the **result** of the function

For instance, consider a function that converts angle to radian:

f: [o,360] ⟶ [0, 2π] such that y = (π/180) * x
where x ∈ [0,360] and y ∈ [0, 2π]

where x [0,360] and y [0, 2π]

Function definition



No *parameters* or **multiple** *parameters* allowed
No *returns* or **multiple** *returns* allowed
Result may be an expression

Function calls

```
def func_name(par 1, par 2, .., par k):
        Function body
        (block of
        instructions)

        return result

res = func_name(arg 1, arg 2, ..., arg k)
```

In general, there is a **positional correspondence** between parameters and arguments.

When a function is called, the **arguments** of the function call are assigned to their corresponding **parameters** of the function definition:

```
def print_twice(text):
  print text
  print text
```

function definition (without return)

```
print_twice("I love cce20003!")    #function call
import math
print_twice(math.pi)     #function call
```

```
import math
def degrees_to_radians(deg):
    rad = (math.pi / 180.0) * deg
return rad

ang = 90
radian = degrees_to_radians(ang)
print radian
```

Function definition (with return)

Function call

```
import math
def degrees_to_radians(deg):
    rad = (math.pi / 180.0) * deg
    return rad
ang = 90
radian = degrees_to_radians(ang)
print radian
```

module

π

Returns the result.

Positional correspondence

```
def compute_interest(amount, rate, years):
    value = amount  *  (1+rate/100.0) ** ye
ars
    return value

amt, r, yrs = 500, 2.0, 15

compute_interest(amt,  r, yrs)
compute_interest(700, 5.0, 30)
```

A **parameter** is a name for an object (a local variable), which can only be used inside the function

BUILT_IN FUNCTIONS AND MODULES

ASTU

Type conversion functions: converting fro m one type to another

```
>>>int(“32”)
32
>>>int(17.3)
17
>>>float(17)
17.0
>>>float(“3.1415”)
3.1415
>>>str(17) + “  “ +  str(3.1415)
‘17  3.1415’
>>>complex(17)
(17 + 0j)
```

Math functions: by importing them from the **math** module:

```
import math
degrees = 45
radians = (math.pi/ 180.0) * degrees
print math.sin(radians)
print math.sqrt(2) / 2
```

When using math functions very often, you can use shorter names:

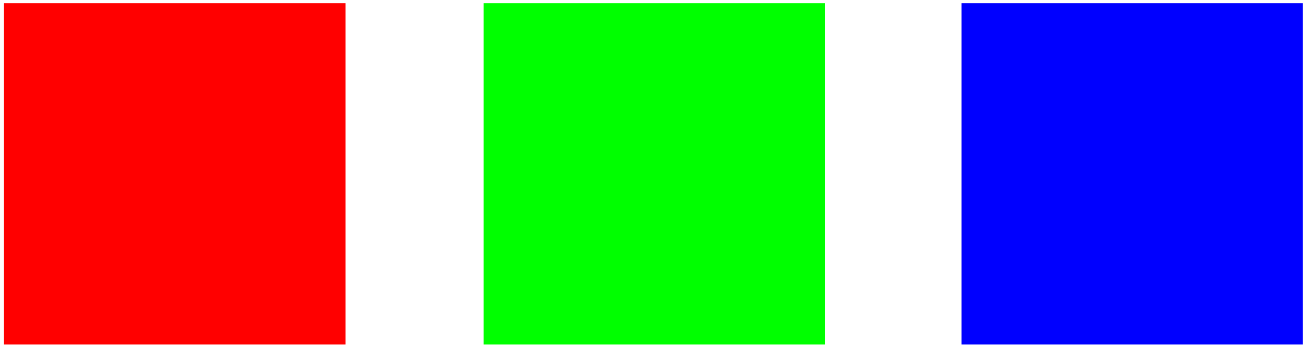
```
import math
sin = math.sin
pi = math.pi
degrees = 45
radians = (pi/180) * degrees
print sin(radians)
```

USER-DEFINED FUNCTIONS

ASTU

From color to intensity(luminance)

(255, 0, 0) (0, 255, 0) (0, 0, 255)



white: (255, 255, 255) black: (0, 0, 0)

```
def luma (p):
    r, g, b = p
    return int(0.213 * r + 0.715 * g + 0.072 * b)
```

Try this function!

```
def blackwhite(img, threshold):
    w, h = img.size()
    for y in range(h):
        for x in range(w):
            v = luma(img.get(x, y))
            if v > threshold:
                img.set(x, y, white)
            else:
                img.set(x, y, black)

from cs1media import *
pict = load_picture("images/yuna.jpg")
```

(r,g,b)

white = (255, 255, 255)
black = (0, 0, 0)

Turning right

```
def turn_right():
    for i in range(3):
        hubo.turn_left()
```

Neither parameters nor returns!

```
s = turn_right()
print s
```

If there is no returns, Python automatically return a special symbol **None**, which represents “nothing”.

Generalization with parameters

```
def turn_right(robot):
    for i in range(3):
        robot.turn_left()

ami = Robot("yellow")

hubo = Robot("blue")

turn_right(ami)

turn_right(hubo)
```

Now, this works for any robots but not just for Hubo!

Absolute value computation

```
def absotute(x):

    if x < 0:
        return –x

    else:
        return x

print
absolute(-7)
```

Multiple returns !

or

```
def absolute():
    if x < 0:
        return –x
    return x

print
absolute(-7)
```

Returning multiple values: A function can return multiple values by returning them as a tuple:

```
def student():
   name = "Hong, Gildo
ng"
   id = 20101234
   return name, id
name1,id1 = student()
   Unpack the tuple!
```

Predicate functions: functions returning a True or False value.

```
def is_divisible(a, b):
    if a % b == 0:
        return True
    else:
        return False
```

or

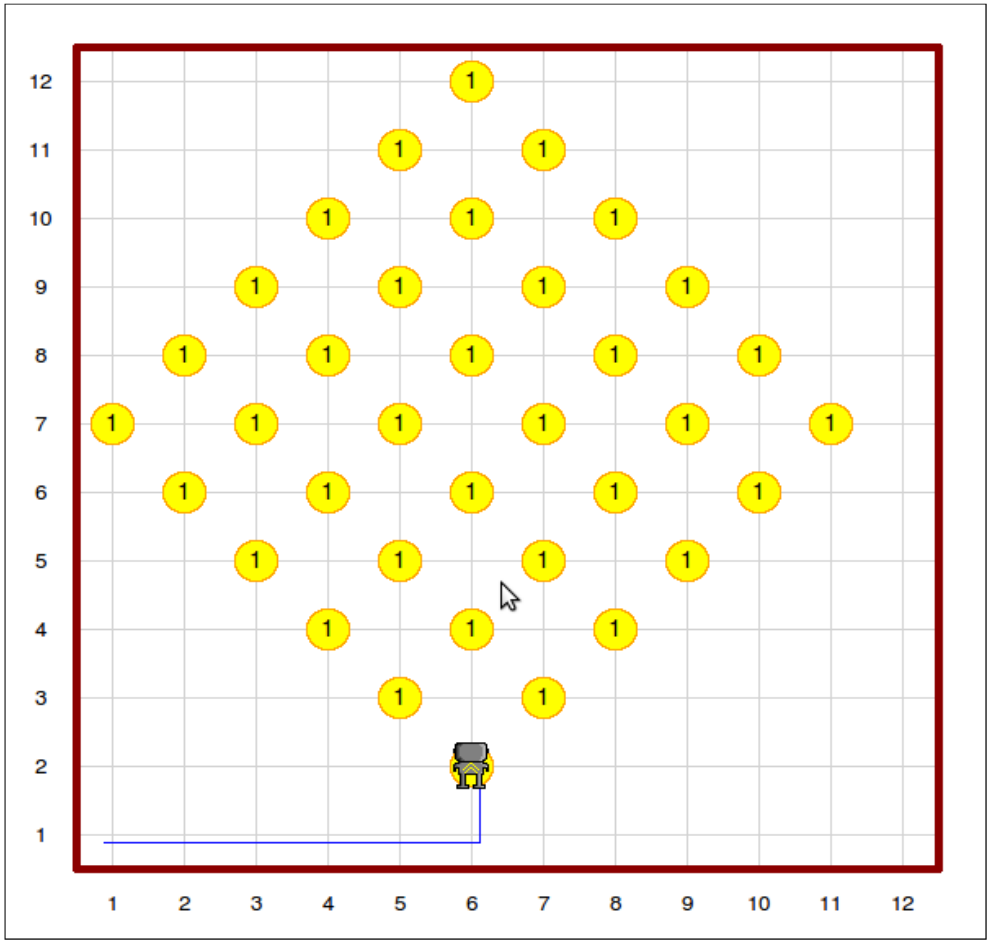
```
def is_divisible(a,b):
    return a % b == 0
```

```
x = 9
y = 3
if is_divisible(x, y):
   print “x is divisible by y.”
```

CASE STUDY: Refining the pseudo code
DECOMPOSITION & ABSTRACTION

ASTU

Harvest revisited



Pseudo code: step 1

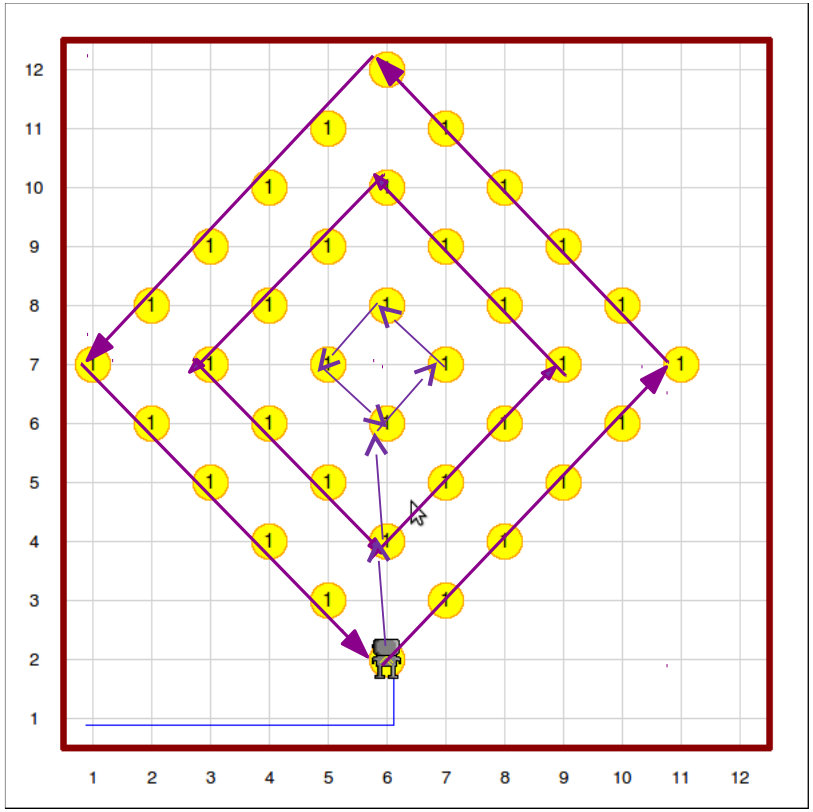
1 Move to the bottom-most point.

2. Pick all beepers

```
def harvest_all(robot):
    move_to_buttom()
```

pick_beepers(robot)

Pseudo-code: step 2



2. Pick up all beepers in the out-most layer
3. Move to the bottom-most position of the middle layer.

4 Pick up all beepers in the middle layer.
5. Move to the bottom-most position in the inner-most layer,

6.Pick up all beepers in the inner-most layer

Refining the pseudo code: Step 2

2.Pick all beepers in the out-most layer.
3.move to the middle layer.

4..Pick all beepers in the middle layer.
5.Move to the inner-most layer.

6.Pick up all beepers in the inner-most layer

2.Pick all beepers in the current layer.
3.move to the next layer.

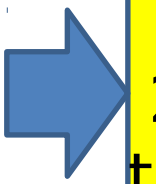
4.Pick all beepers in the current layer.
5.Move to the next layer.

6.Pick up all beepers in the current layer

2.Pick all beepers in the current layer.
3.move to the next layer.

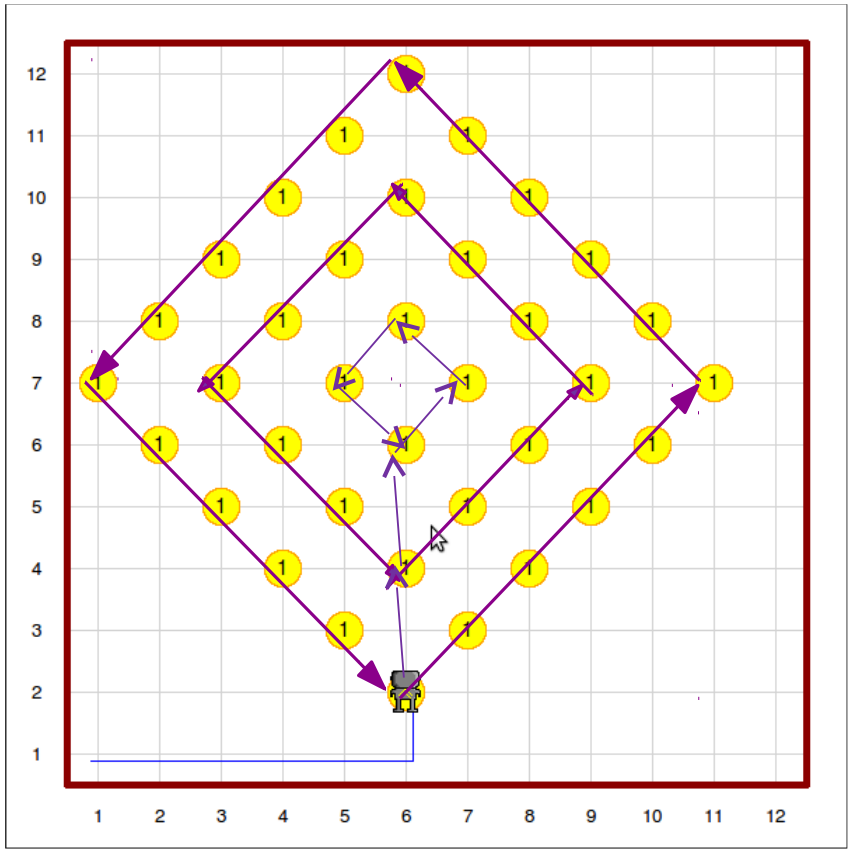
4.Pick all beepers in the current layer.
5.Move to the next layer.

6.Pick up all beepers in the current layer



2. Repeat the following steps three times:
2.1. Pick all beepers in the current layer.
2.2. If not the inner-most layer, move to the next layer.

What is the main difference between layers?



The number of beepers / side!

How to compute it ?

```
5 – 2 * i  for i = 0, 1, 2
```

Why? (your is creativity needed!!)

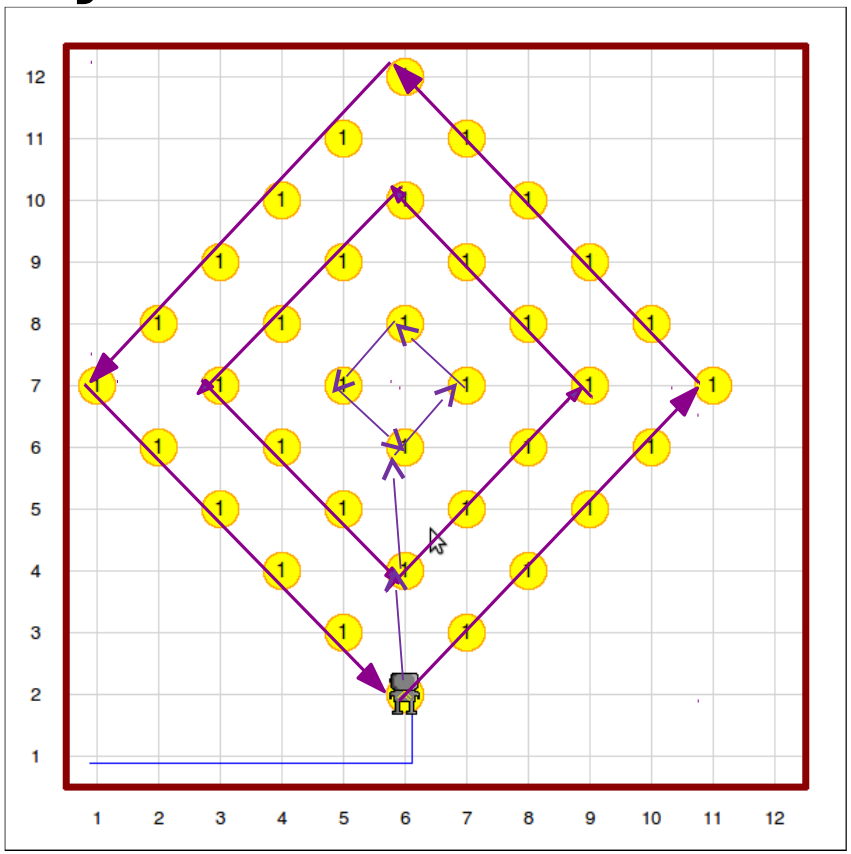
---> Good algorithm

2. Repeat the following steps three times:
2.1. Pick all beepers in the current layer.
2.2. If not the inner-most layer, move to the next layer.

```
def pick_beepers(robot):
    for i in range(3):
        n = 5 - 2 * i
        diamond(robot, n)
        if n > 1 :
            robot.move()

robot.move()
```

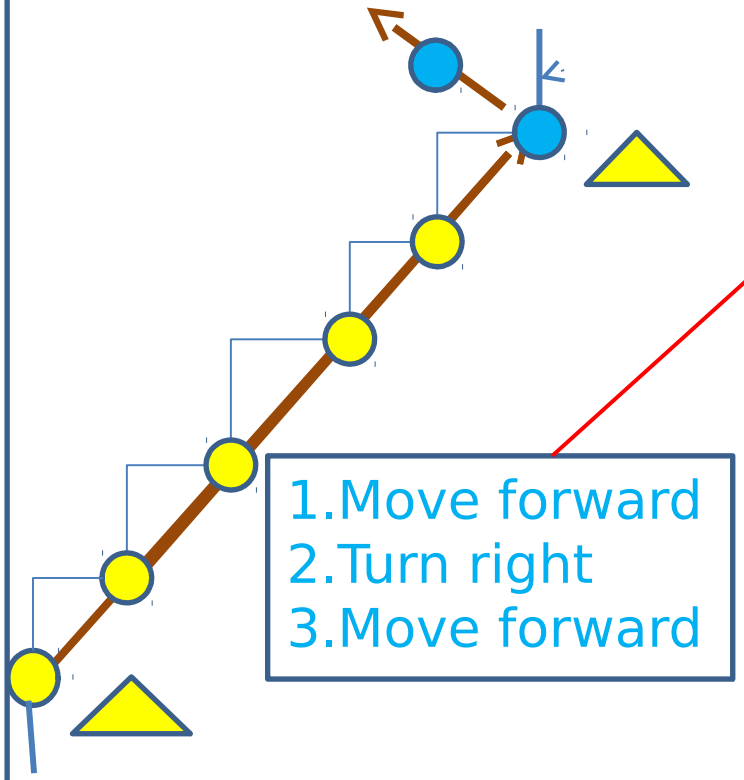
2.1 How to pick-up all beepers in the current l ayer



2.1 For each of four sides, do the followings:
2.1.1 Pick up all beepers in the current side.
2.1.2 prepare for moving to the next side.

```
def diamond(robot,n):
    for i in range(4):
        move_and_pick()
        for_next_side()
```

2.1.1 How to pick the beepers in the current side



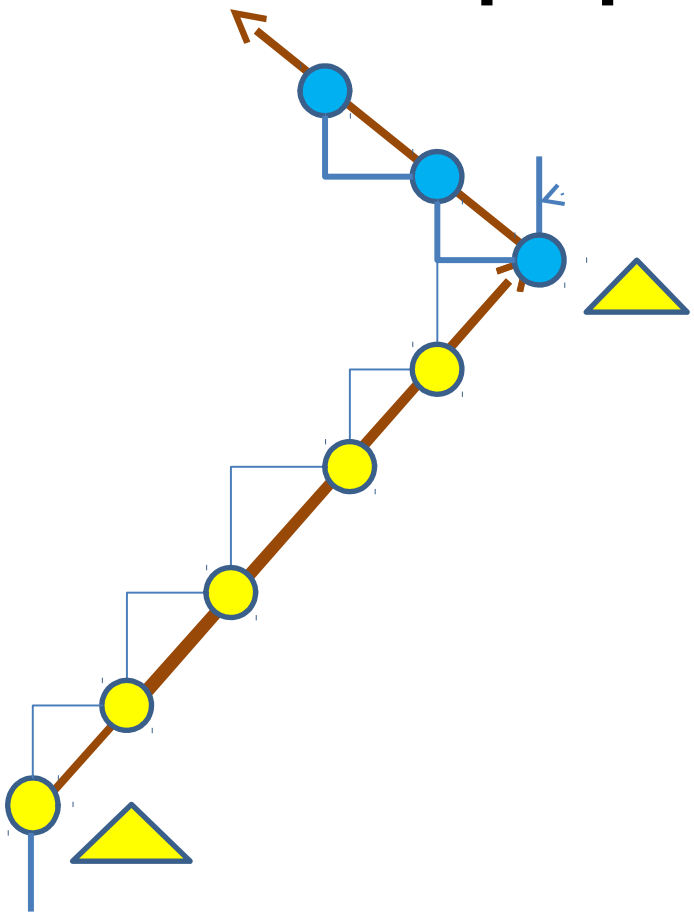
1.Move forward
2.Turn right
3.Move forward

At every position on a side:

1.Pick up a beeper.
2.Go up a stair.
3.Prepare for next stair

```
def pick_and_move(robot, n):
    for i in range(n):
        robot.pick_beeper()
        robot.move()
        turn_right(robot)
        robot.move()
```

2.1.2 How to prepare for the next side



Turn left !

~~def for_next_side(robot):~~
~~   robot.turn_left()~~

```
def diamond(robot,n):
    for i in range(4):
        move_and_pick()
        for_next_side()

    robot.turn_left()
```

Program

```
from cs1robots import *
load_world("worlds/harvest3.wld")
hubo = Robot()
Put function definitions, here.


harvest_all(hubo)
```

KEYBOARD INPUT

ASTU

```
name = raw_input("What is your name? ")

print "Welcome to CCE20003, " + na
me

raw_n = raw_input("Enter a positiv
e integer> ")

n = int(raw_n)   Why?

for i in range(n):

   print "*" * i
```

Argument: a prompt displayed on the m
onitor.