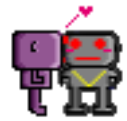
Introduction to Programming
CS101

Spring 2012

Lecture #4



Functions

The name function comes from mathematics. A function is a mapping from one set to another set:

$$f : \mathbb{R} \to \mathbb{R}$$

$$x \mapsto \pi \times \frac{x}{180.0}$$

Here, x is the argument of the function, $f(x)$ is the result of the function.

In Python, functions also take arguments and return a result:

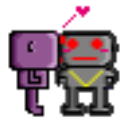
```
def to_radians(deg):
  return (deg / 180.0) * math.pi

>>> a = to_radians(90)
>>> print a
1.5707963267948966
```

Useful functions

Python comes with many built-in functions.

Type conversion functions convert from one type to another type:

```
>>> 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)
```



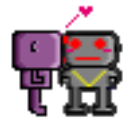
Math functions

To use math functions, we need to tell Python that we want to use the `math` module:

```
import math
degrees = 45
radians = degrees / 360.0 * 2 * math.pi
print math.sin(radians)
print math.sqrt(2) / 2
```

When using math functions often, we can use shorter names:

```
import math
sin = math.sin
pi = math.pi
radians = degrees / 360.0 * 2 * pi
print sin(radians)
```



Defining functions with parameters

The function definition uses names for the arguments of the function. These names are called parameters:

```
def compute_interest(amount, rate, years):
```

Inside the function, the parameter is just a name:

```
    value = amount * (1 + rate/100.0) ** years
```

When we have computed the result of the function, we return it from the function. The function ends at this point, and the result object is given back:

```
    return value
```

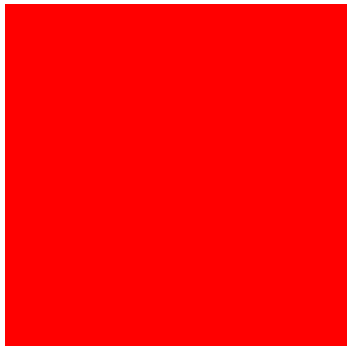
We can now call the function with different argument values:

```
>>> s1 = compute_interest(200, 7, 1)
>>> s2 = compute_interest(500, 1, 20)
```

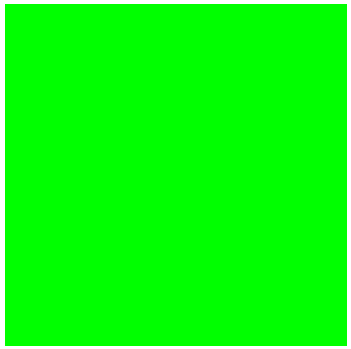
Converting to black-and-white

What is the light intensity (luminance) of pixel `(r,g,b)`?

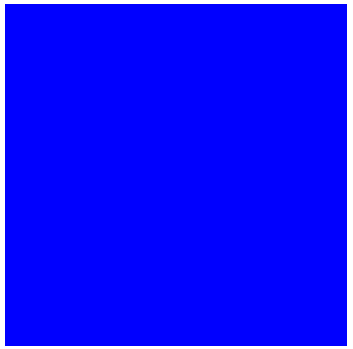
(255, 0, 0)



(0, 255, 0)

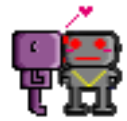


(0, 0, 255)



A good formula is:

```
def luminance(p):
  r, g, b = p
  return int(0.299 * r + 0.587 * g + 0.114 * b)
```



More than one return in a function

Compute the absolute value (like builtin function `abs`):

```
def absolute(x):
  if x < 0:
    return -x
  else:
    return x
```

The same function can be written like this:

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

But not like this:

```
def absolute(x):
  if x < 0:
    return -x
  if x > 0:
    return x
```

Returning a boolean

A function that tests a condition and returns either `True` or `False` is often called a predicate:

```
# is integer a divisible by b?
def is_divisible(a, b):
  if a % b == 0:
    return True
  else:
    return False
```

Easier:

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

A predicate (function) can be used directly in an `if` or `while` statement:

```
if is_divisible(x, y):
  print 'x is divisible by y'
```

Functions without results

We have seen many functions that do not use `return`:

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

In fact, a function that does not call `return` automatically returns `None`:

```
>>> s = turn_right()
>>> print s
None
```

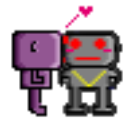
Calling functions

When a function is called, the arguments of the function call are assigned to the parameters:

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

The number of arguments in the function call must be the same as the number of parameters.

```
>>> print_twice("I love CS101")
I love CS101
I love CS101
>>> print_twice(math.pi)
3.14159265359
3.14159265359
```



Hubo's family

We can now write a `turn_right` function that will work for any robot, not just for Hubo:

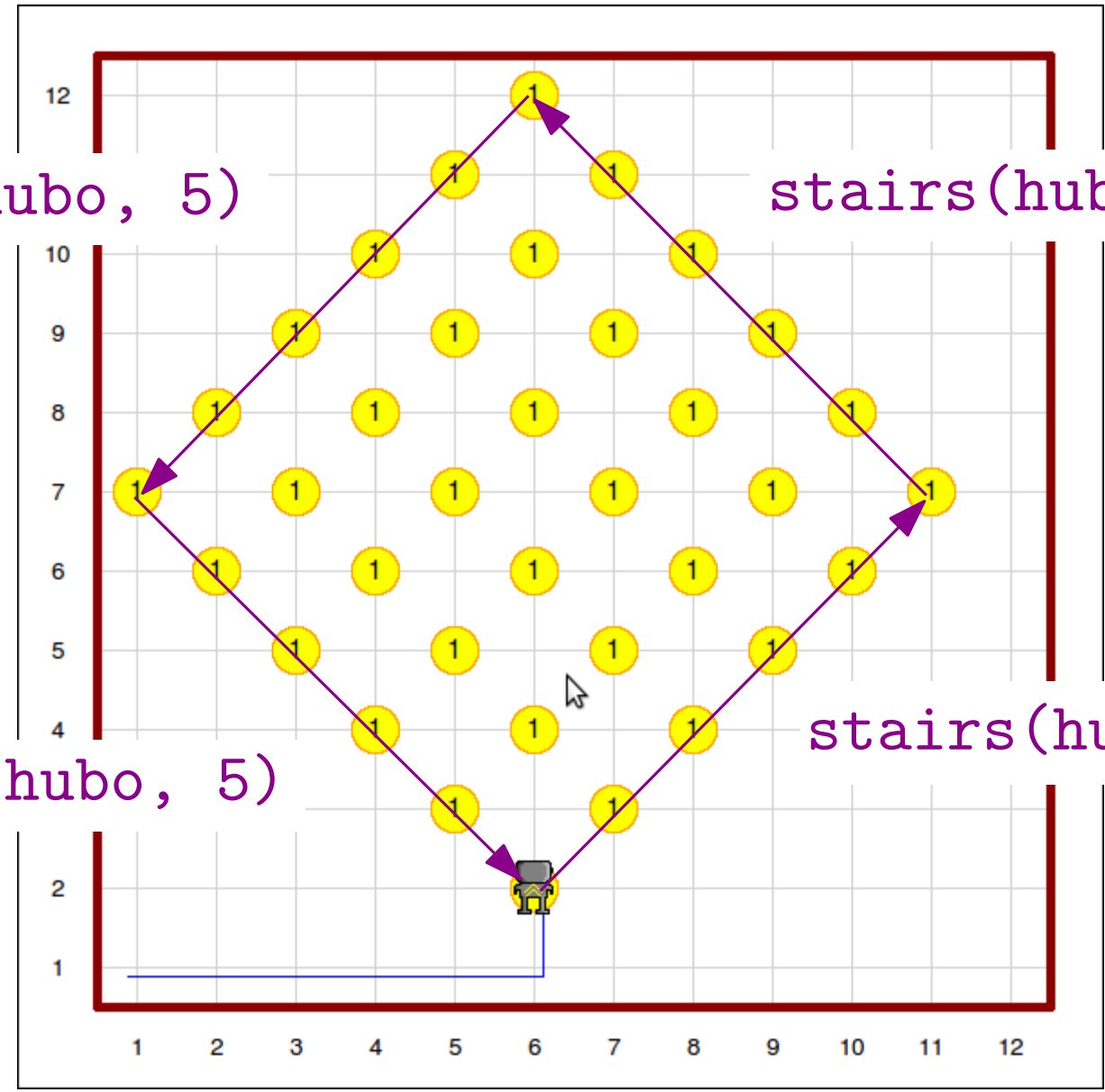
```
def turn_right(robot):
  for i in range(3):
    robot.turn_left()

ami = Robot("yellow")
hubo = Robot("blue")
turn_right(ami)
turn_right(hubo)
```

Remember: A parameter is a name for an object. The name can only be used inside the function.

Harvesting again

```
stairs(hubo, 5)
```

```
stairs(hubo, 5)
```




```
stairs(hubo, 5)
```

```
stairs(hubo, 5)
```

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

def diamond(robot, n):
  for i in range(4):
    stairs(robot, n)
    robot.turn_left()
```

```
def harvest_all(robot):
  for i in range(3):
    n = 5 - 2 * i
    diamond(robot, n)
    hubo.move()
    hubo.move()
```

Converting to black and white, again

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

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

pict = load_picture("../photos/yuna1.jpg")
blackwhite(pict, 100)
pict.show()
```

Returning more than one value

A function can only return one value.

But this value can be a tuple, and functions can return arbitrarily many values by returning them as a tuple:

```
def student():
  name = "Hong, Gildong"
  id = 20101234
  return name, id
```

Often function results are unpacked immediately:

```
name, id = student()
```

Keyboard input

The `raw_input` function waits for the user to enter a string on the keyboard. When the user presses the Enter key, the whole string is returned:

```
name = raw_input("What is your name? ")
print "Welcome to CS101, " + name
```

If we need a number, we should convert the string:

```
raw_n = raw_input("Enter a positive integer> ")
n = int(raw_n)
for i in range(n):
  print "*" * i
```