Programming using Python

f(unctions)

BY

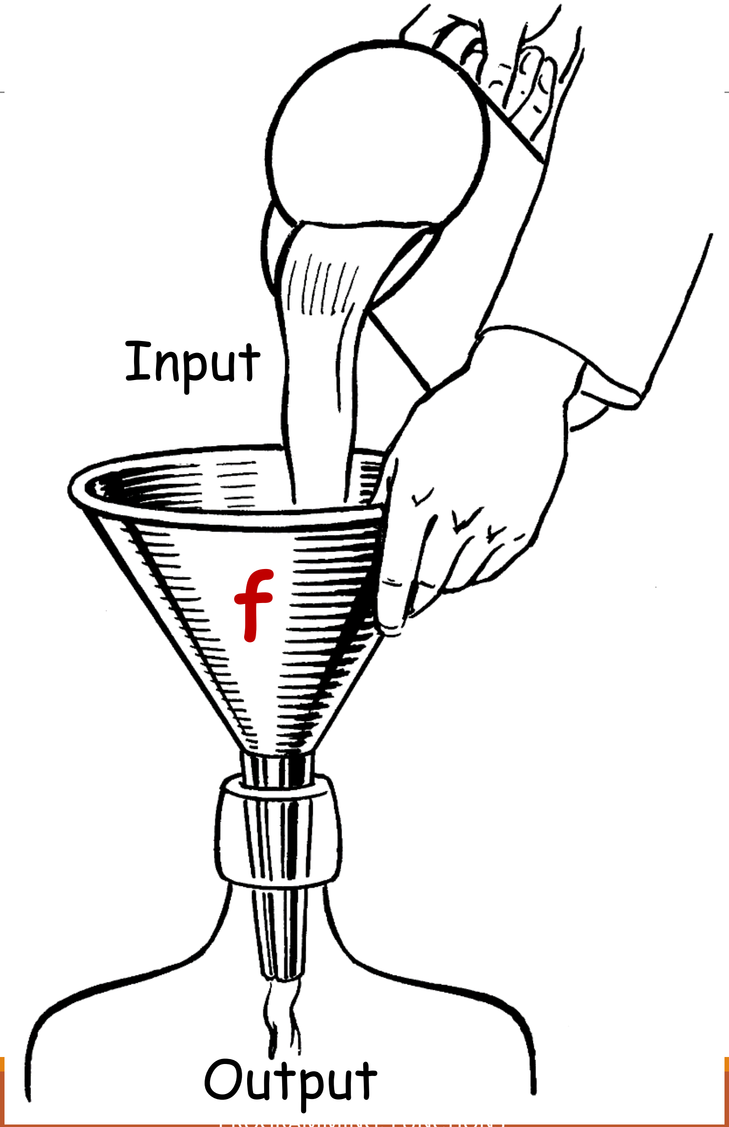
MR. SUMEDRAO M. GAIKWAD

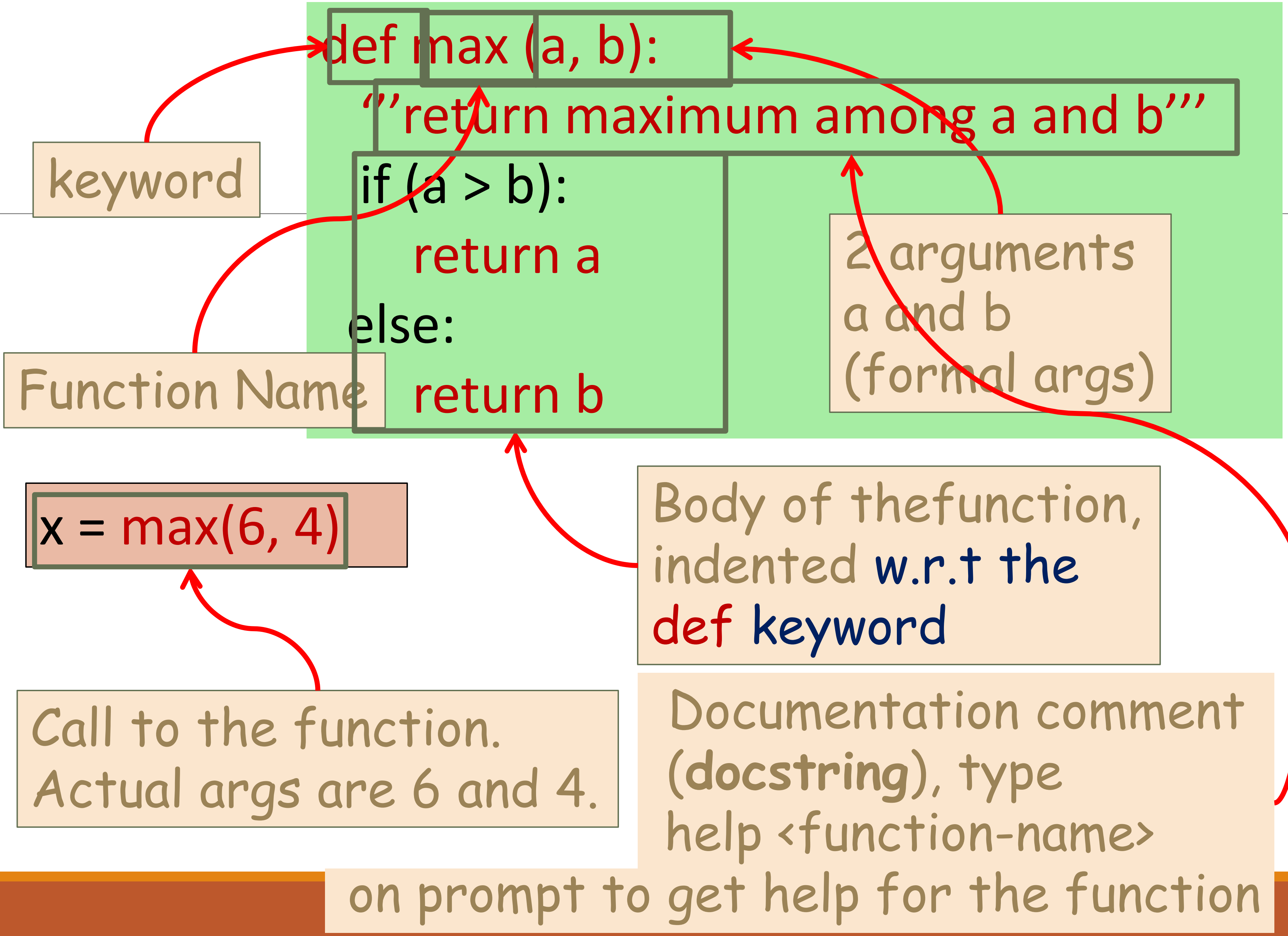
BCA DEPARTMENT

VIVEKANAND COLLEGE(AUTONOMOUS)

KOLHAPUR

Parts of a function





```
def max (a, b):
  '''return maximum among a and b'''
  if (a > b):
     return a
  else:
     return b
```

```
x = max(6, 4)
```

keyword

Function Name

2 arguments a and b (formal args)

Body of thefunction, indented **w.r.t the def keyword**

Call to the function. Actual args are 6 and 4.

Documentation comment (**docstring**), type help <function-name> on prompt to get help for the function

```
def max (a, b):
   “‘‘return maximum among a and b’’’
   if (a > b):
       return a
  else:
       return b
```

```
In[3] : help(max)
Help on function max in module __main__:

max(a, b)
    return maximum among a and b
```

Keyword Arguments

```
def printName(first, last, initials) :
  if initials:
    print (first[0] + '. ' + last[0] + '.')
  else:
    print (first, last)
```

Note use of [0] to get the first character of a string. More on this later.

Call	Output
printName('Acads', 'Institute', False)	Acads Institute
printName('Acads', 'Institute', True)	A. I.
printName(last='Institute', initials=False, first='Acads')	Acads Institute
printName('Acads', initials=True, last='Institute')	A. I.

Keyword Arguments

Parameter passing where formal is bound to actual using formal's name

Can mix keyword and non-keyword arguments

- All non-keyword arguments precede keyword arguments in the call
- Non-keyword arguments are matched by position (order is important)
- Order of keyword arguments is not important

Default Values

```
def printName(first, last, initials=False) :
  if initials:
    print (first[0] + '. ' + last[0] + '.')
  else:
    print (first, last)
```

Note the use of "default" value

Call	Output
printName('Acads', 'Institute')	Acads Institute
printName(first='Acads', last='Institute', initials=True)	A. I.
printName(last='Institute', first='Acads')	Acads Institute
printName('Acads', last='Institute')	Acads Institute

Default Values

Allows user to call a function with fewer arguments

Useful when some argument has a fixed value for most of the calls

All arguments with default values must be at the end of argument list

- non-default argument can not follow default argument

Globals

Globals allow functions to communicate with each other indirectly

- Without parameter passing/return value

Convenient when two seemingly “far-apart” functions want to share data

- No *direct* caller/callee relation

If a function has to update a global, it must re-declare the global variable with global keyword.

Globals

```
PI = 3.14
def perimeter(r):
   return 2 * PI * r
def area(r):
   return PI * r * r
def update_pi():
   global PI
   PI = 3.14159
```

```
>>> print(area (100))
31400.0
>>> print(perimeter(10))
62.800000000000004
>>> update_pi()
>>> print(area(100))
31415.999999999996
>>> print(perimeter(10))
62.832
```

defines PI to be of float type with value 3.14. PI can be used across functions. Any change to PI in update_pi will be visible to all due to the use of global.