UNIVERSITY OF MARYLAND
ROBERT H. SMITH SCHOOL OF BUSINESS

Essential Data Skills for Business Analytics

Lecture 6 : Functions and Parameters

Decision, Operations & Information Technologies
Robert H. Smith School of Business
Spring, 2020

Reusable codes

UNIVERSITY OF MARYLAND
ROBERT H. SMITH SCHOOL OF BUSINESS

calc:

Do the following operations for two numbers and print results:

+, -, *, /

```
def calc(a, b):
        sum = a + b
        subtraction = a – b
        multiply = a * b
        division = a / b
        print (“sum is: ”, sum)
        print (“subtraction is: ”, subtraction)
        print (“multiplication is: ”, multiply)
        print (“division is: ”, division)
```

- >>> calc (5,7)
- >>> calc (3.0, 4)
- >>> calc (1.2, 2.5)

Function

- A function is a block of organized, reusable code that is used to perform a group of related actions.

- Functions provide better modularity for your application and a high degree of code reusing.

Python functions

- There are two kinds of functions in Python
  - Build-in functions that are provided as part of Python
    - input(), type(), float(), int(), …
  - Functions that we define ourselves and then use

Build-in functions

- Math functions
  - Python has a math module that provides most of the familiar mathematical functions.
  - Before use all these functions, we have to import them: >>> import math
  - To call one of the functions, we have to specify the name of the module and the name of the function, separated by a dot. This format is also called **dot notation**.
    - >>>decibel = math.log10(17.0)
    - >>>angle = 1.5
    - >>>height = math.sin(angle)

Math function examples


```
>>> degrees = 45
>>> angle = degrees * 2 * math.pi / 360
>>> math.sin(angle)
0.707106781187
```

- Combination of functions

```
>>> x = math.exp(math.log(10))
```

Many other math functions

UNIVERSITY OF MARYLAND
ROBERT H. SMITH SCHOOL OF BUSINESS

Function name	Description
`abs(`**value**`)`	absolute value
`ceil(`**value**`)`	rounds up
`cos(`**value**`)`	cosine, in radians
`degrees(`**value**`)`	convert radians to degrees
`floor(`**value**`)`	rounds down
`log(`**value**`, `**base**`)`	logarithm in any base
`log10(`**value**`)`	logarithm, base 10
`max(`**value1**`, `**value2**`, ...)`	larger of two (or more) values
`min(`**value1**`, `**value2**`, ...)`	smaller of two (or more) values
`radians(`**value**`)`	convert degrees to radians
`round(`**value**`)`	nearest whole number
`sin(`**value**`)`	sine, in radians
`sqrt(`**value**`)`	square root
`tan(`**value**`)`	tangent

Constant	Description
`e`	2.7182818...
`pi`	3.1415926...

Defining our own functions

- Simple rules to define a function
  1. Function blocks begin with the keyword **def** followed by the function name, parentheses (), and a colon :
  2. Any parameters or arguments should be placed within these parentheses.
  3. The code block within every function is indented.

```
def function_name (parameters):
    statement 1 in the function body
    statement 2 in the function body
    …
    statement n in the function body
```

Building our own functions

- Defining a function doesn't mean we execute the body of the function
- test.py file:

```
x = 5
print ('Hello')

def print_oks():
    print ('OK 1')
    print ('OZ 2')

print 'Yo'
x = x + 2
print (x)
```

- python test.py

```
Hello
Yo
7
```

Calling a function

- Once we have defined a function, we can call (or invoke) it as many times as we like
- This is a store and reuse pattern
- test.py file:

```
x = 5                           (1)
print ('Hello')                 (2)

def print_oks():                (5)
    print ('OK 1')              (6)
    print ('OZ 2')              (7)

print ('World')                 (3)
print_oks()                     (4)
x = x + 2                       (8)
print (x)                       (9)
```

- python test.py

```
Hello
World
OK 1
OK 2
7
```

Arguments

- An argument is a value we pass into the function as its input when we call the function
- Sometimes, the function doesn't need input (no arguments)
- We use arguments so we can direct the function to do different kinds of work when we call it at different times
- We put the arguments in parentheses after the name of the function

```
def calc(a, b):
        sum = a + b
        subtraction = a – b
        multiply = a * b
        division = a / b
        print (“sum is: ”, sum)
        print (“subtraction is: ”, subtraction)
        print (“multiplication is: ”, multiply)
        print (“division is: ”, division)
```

Arguments


```
def calc(a, b):
        sum = a + b
        subtraction = a – b
        multiply = a * b
        division = a / b
        print (“Sum is: ”, sum)
        print (“Subtraction is: ”, subtraction)
        print (“Multiplication is: ”, multiply)
        print (“Division is: ”, division)
```

```
x = 3                                   (1)
y = 5                                   (2)
m = 2.0                                 (3)
n = 7                                   (4)
i = 2.5                                 (5)
j = 2.2                                 (6)
calc(x, y)      ## or calc(3, 5)        (7)
calc(m, n)      ## or calc(2.0, 7)      (8)
calc(i, j)      ## or calc(2.5, 2.2)    (9)
```

Arguments

Outputs:

```
Sum is: 8
Subtraction is: -2
Multiplication is: 15
Division is: 0
Sum is: 9.0
Subtraction is: -5.0
Multiplication is: 14.0
Division is: 0.2857142857142857
Sum is: 4.7
Subtraction is: 0.3
Multiplication is: 5.5
Division is: 1.1363636363636362
```

Parameters

- A parameter is a variable which we use in the function definition
- It is a "**handle**" that allows the code in the function to access the arguments for a particular function invocation

Parameter

```
def greet(lang):
    if lang == 'es':
        print ('Hola')
    elif lang == 'fr':
        print ('Bonjour')
    else:
        print ('Hello')

>>> greet('en')
Hello
>>> greet('es')
Hola
>>> greet('fr')
Bonjour
>>>
```

Back to arguments

- In Python, there are four types of arguments
  - ❑Required arguments
  - ❑Keyword arguments
  - ❑Default arguments
  - ❑Variable-length arguments

Required arguments

UNIVERSITY OF MARYLAND
ROBERT H. SMITH SCHOOL OF BUSINESS

- Required arguments are the arguments passed to a function in correct positional order.
- **The number of arguments in the function call should match exactly with the parameters in the function definition.**

```
#!/usr/bin/python

# define a function
def printme( s ):
        print (s)
        return
# call the function
printme()                    → printme("I'm KZ")
```

```
Traceback (most recent call last):
   File "test.py", line 8, in <module>
        printme();
TypeError: printme() takes exactly 1 argument (0 given)
```

Required argument

Keyword arguments

UNIVERSITY OF MARYLAND
ROBERT H. SMITH SCHOOL OF BUSINESS

- Keyword arguments are related to the function calls.
- When you use keyword arguments in a function call, the caller identifies the arguments by the parameter name.
- This allows you to skip arguments or place them out of order because the Python interpreter is able to use the keywords provided to match the values with parameters.

```
#!/usr/bin/python

# Function definition is here
def printinfo( name, age ):
   "This prints a passed info into this function"
   print "Name: ", name
   print "Age ", age
   return;

# Now you can call printinfo function
printinfo( age=50, name="miki" )
```

Keyword arguments

```
Name:  miki
Age:  50
```

Required arguments:
printinfo("miki", 50)

Printinfo(50, "miki")

Default arguments

UNIVERSITY OF MARYLAND
ROBERT H. SMITH SCHOOL OF BUSINESS

- A default argument is an argument that assumes a default value if a value is not provided in the function call for that argument.

```
#!/usr/bin/python

# Function definition is here
def printinfo( name, age = 35 ):
   "This prints a passed info into this function"
   print "Name: ", name
   print "Age ", age
   return;

# Now you can call printinfo function
printinfo( age=50, name="miki" )
printinfo( name="miki" )
```

Keyword arguments

Default arguments

```
Name:  miki
Age  50
Name:  miki
Age  35
```

Variable-length arguments

UNIVERSITY OF MARYLAND
ROBERT H. SMITH SCHOOL OF BUSINESS

- You may need to process a function for more arguments than you specified while defining the function.
- Variable-length arguments are not named in the function definition.
- Use * before the variable name that holds the values of all other variable arguments in the definition.
- This variable remains empty if no additional arguments are specified during the call.

```
def functionaame(< formal_args >, *var_args_tuple):
    statements
```

```
#!/usr/bin/python

# Function definition is here
def printinfo( arg1, *vartuple ):
   "This prints a variable passed arguments"
   print "Output is: "
   print arg1
   for var in vartuple:
      print var
   return;

# Now you can call printinfo function
printinfo( 10 )
printinfo( 70, 60, 50 )
```

```
Output is:
10
Output is:
70
60
50
```

Anonymous functions

- Anonymous functions are not declared in the standard manner by using the **def** keyword.

- Use the **lambda** keyword to create small anonymous functions.

```
lambda : expression
lambda arg1, arg2, ..., argN: expression
```

Anonymous function rules (1)

- Lambda forms can take any number of arguments but return just one value in the form of an expression. They can not contain commands or multiple expression.

```
>>>func = lambda : 5+3
>>>func
8

>>>func = lambda a, b: a+b
>>>sum = func(5,3)
>>>print ("sum is: ", sum)
sum is: 8
```

```
>>>func = lambda a,b: a+b; a-b
>>>func(5,3)
8
```

Anonymous function rules (2)

- An anonymous function can not be a direct call to print because lambda requires an expression.

```
>>>func = lambda : print "hello world"
>>>func
```

The *return* statement

UNIVERSITY OF MARYLAND
ROBERT H. SMITH SCHOOL OF BUSINESS

- Often a function will take its arguments, do some computation, and return a value to be used as the **value of the function call** in the calling expression.
- The ***return*** keyword is used for this.
- A return statement with no arguments is the same as return none.

```
def greet():
    return "Hello"          (2) (4)

print greet(), "Glenn"      (1)
print greet(), "Sally"      (3)
```

```
Hello Glenn
Hello Sally
```

The *return* statement

UNIVERSITY OF MARYLAND
ROBERT H. SMITH SCHOOL OF BUSINESS

- The ***return*** statement ends the function execution and "sends back" the result of the function

```
def greet(lang):
    if lang == 'es':
        return 'Hola'
    elif lang == 'fr':
        return 'Bonjour'
    else:
        return 'Hello'

>>> print (greet('en'),'Glenn')
Hello Glenn
>>> print (greet('es'),'Sally')
Hola Sally
>>> print (greet('fr'),'Michael')
Bonjour Michael
```

```
What greet does:

1)Receives the argument value

2)If the lang is es, then
return Hola, otherwise,

2.1)If the lang is fr, then
return Bonjour, otherwise
return Hello
```

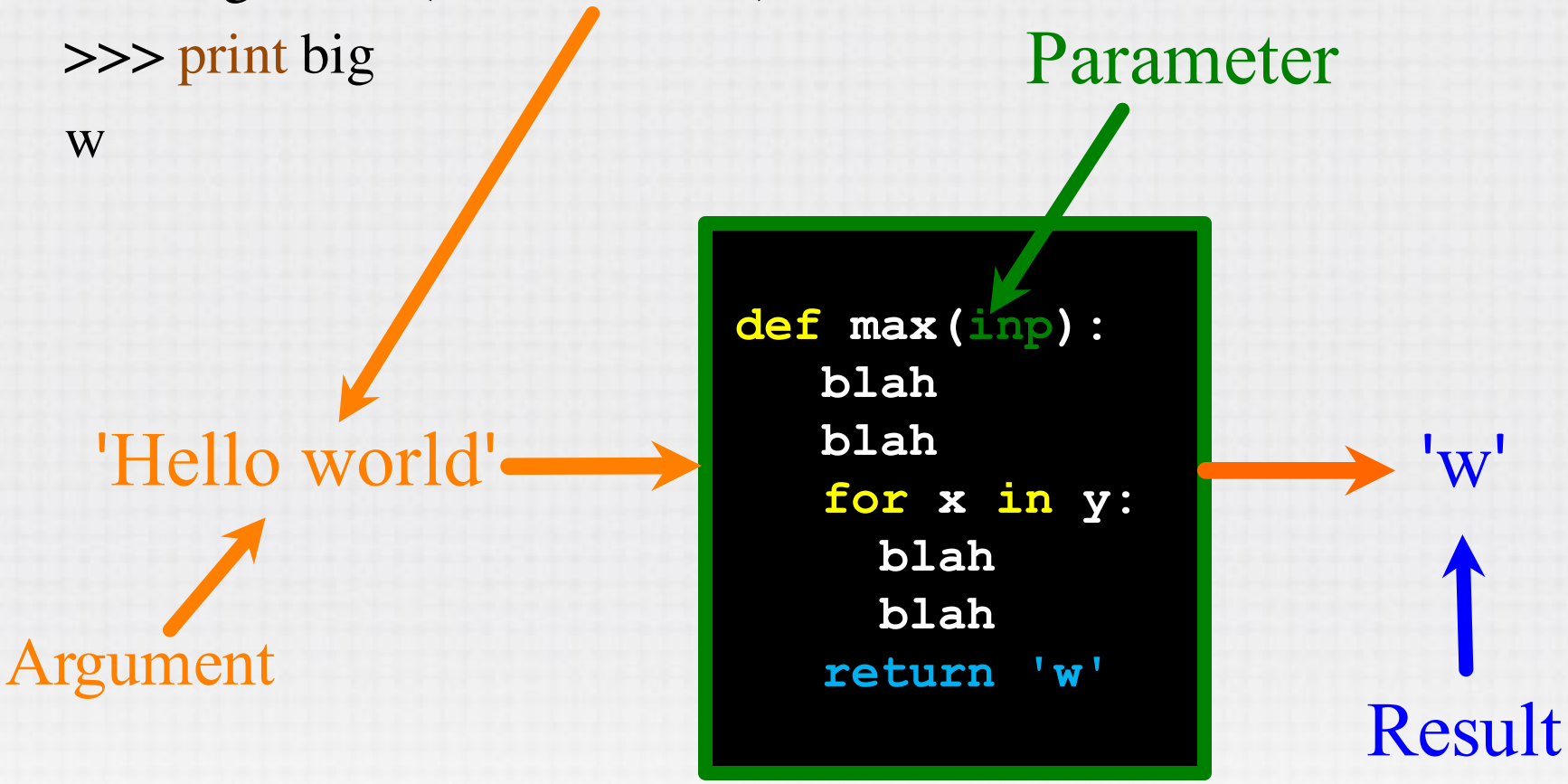
An example


```
#!/usr/bin/python

# Function definition is here
def sum( arg1, arg2 ):
   # Add both the parameters and return them."
   total = arg1 + arg2
   print "Inside the function : ", total
   return total;

# Now you can call sum function
total = sum( 10, 20 );
print "Outside the function : ", total
```

```
Inside the function :  30
Outside the function :  30
```

Arguments, parameters, and results


```
>>> big = max('Hello World')
>>> print big
w
```



Multiple parameters / arguments

UNIVERSITY OF MARYLAND
ROBERT H. SMITH SCHOOL OF BUSINESS

- We can define more than one parameter in the function definition
- We simply add more arguments when we call the function
- We match the number and the order of arguments and parameters

```
def addtwo(a, b):
    added = a + b
    return added

x = addtwo(3, 5)
print (x)
```

```
8
```

Void (non-fruitful) functions

- When a function does not return a value, we call it a "void" function
- Functions that return values are "fruitful" functions
- Void functions are "not fruitful"

Program development

- To deal with increasingly complex programs, we are going to suggest a technique called incremental development.
- It is to avoid long debugging sessions by adding and testing only a small amount of code at a time.

$$\text{distance} = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

- The first step is to consider what a distance function should look like in Python.

Program development

$$\text{distance} = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

- What are the inputs (parameters) and what is the output (return value)?

```
def distance(x1,y1,x2, y2):
    return 0.0
```

- To test the function, we call it with sample values

```
>>> distance(1,2,4,6)
0
```

Program development

UNIVERSITY OF MARYLAND
ROBERT H. SMITH SCHOOL OF BUSINESS

- Find the differences x2-x1 and y2-y1

```
def distance(x1,y1,x2, y2):
    dx = x2-x1
    dy = y2-y1
    print ('dx is:' , dx)
    print ('dy is:' , dy)
    return 0.0
```

- To test the function, we call it with sample values

```
>>> distance(1,2,4,6)
dx is: 3
dy is: 4
0
```

Program development

- Compute the sum of squares of dx and dy

```
def distance(x1,y1,x2, y2):
    dx = x2-x1
    dy = y2-y1
    dsquared = dx**2 + dy**2
    result = math.sqrt(dsquared)
    return result
```

Recursion

- It is legal for one function to call another.

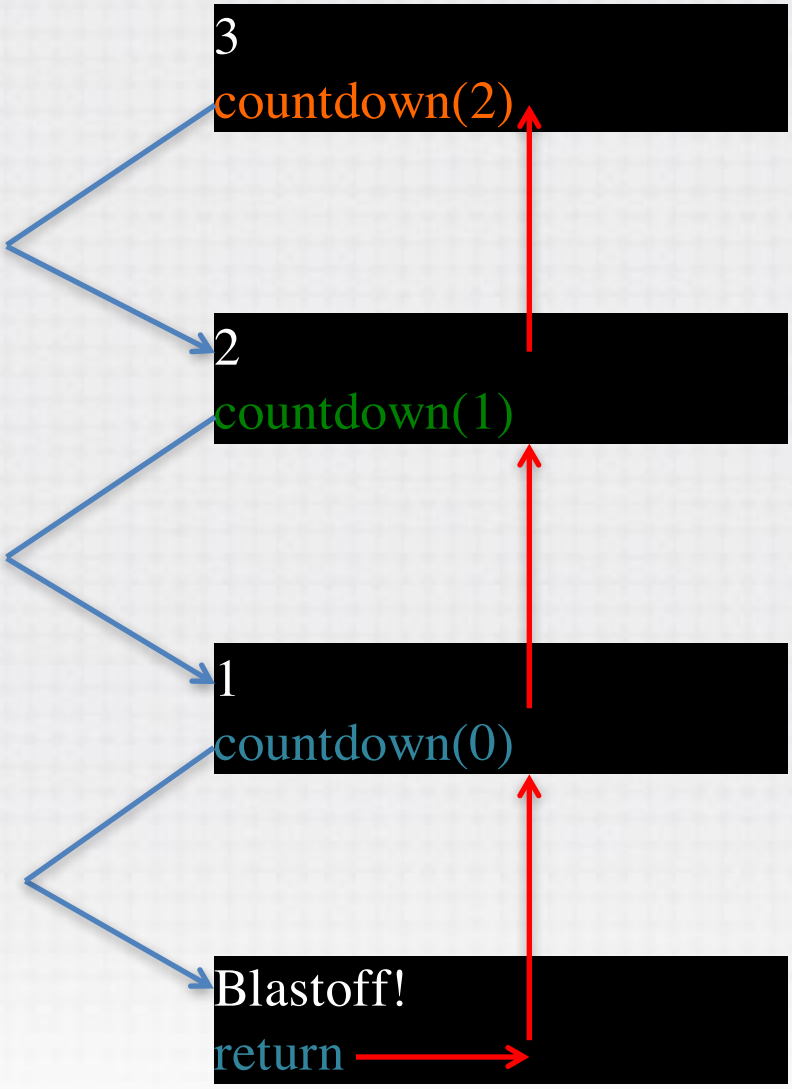
```
def distance(x1, y1, x2, y2):
    dx = subtraction(x2,x1)
    dy = subtraction(y2,y1)
    dsquared = dx**2 + dy**2
    result = math.sqrt(dsquared)
    return result

def subtraction(a, b):
    sub = a-b
    return sub
```

- What about calling itself?

Recursion

```
def countdown(n):
    if n == 0:
        print "Blastoff!"
    else:
        print n
        countdown(n-1)
```

```
>>> countdown(3)
```



Examples


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

>>> factorial(4)
```

```
def fibonacci(n):
    if n == 0:
        return 0
    elif n == 1:
        return 1
    else:
        return fibonacci(n-1) + fibonacci(n-2)

>>> fibonacci(5)
```