IDCE 302: Chapter 5

Fruitful Functions

Jie Tian, PhD

Clark University

Outline

- Fruitful Function
- Incremental Development Strategy
- More Recursion
- Demos

Return Values

Fruitful functions are those functions that return values.

Example 1:

```
  import math

# circle-area calculation
def calarea(radius):
    area = math.pi * radius**2
    return area

r=1
print "The area of a circle with a radius of", r, ”is ", calarea(r)
r=2
print "The area of a circle with a radius of", r, ”is ", calarea(r)
r=3
print "The area of a circle with a radius of", r, ”is ", calarea(r)
```

Example 2:

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

  val = -3
  print "The absolute value of ", val, " is ", absoluteValue(val)
  val = 2
  print "The absolute value of ", val, " is ", absoluteValue(val)
  val = 0
  print "The absolute value of ", val, " is ", absoluteValue(val)
```

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

Incremental Development

```
import math

"""
Input parameters:
    x1, y1: input coordinate; the first point.
    x2, y2: input coordinate; the second point.
Return value:
    Euclidean distance of the two input points
    D = sqrt(dX^2 + dY^2)
"""
def distance(x1, y1, x2, y2):
    return math.sqrt((x2-x1)**2+(y2-y1)**2)

# call distance function with two points' coordinates: x1,
y1, x2, y2
print distance(0,0, 3,4)
```

When dealing with a complex problem, **we do it one-step a time!**

```
import math
"""
Input parameters:
   x1, y1: input coordinate; the first point.
   x2, y2: input coordinate; the second point.
Return value:
   Euclidean distance of the two input points
   D = sqrt(dX^2 + dY^2)
"""
def distance(x1, y1, x2, y2):
   # temporary variables are used to make debugging easier
   dx = x2 - x1
   dy = y2 - y1
   dsquared = dx*dx + dy*dy  # you may use dx**2 + dy**2
   result = math.sqrt(dsquared)
   return result # the two lines can be combined.

# call distance function with two points' coordinates: (0,0) and (3,4)
print distance(0, 0, 3, 4)
```

Composition

- Call one function from within another function

```
import math
"""
Input parameters:
    cx, cy: center of a circle.
    px, py: a point on the perimeter
Return value:  area of the circle
"""
def calcArea(cx, cy, px, py):
    radius = distance(cx, cy, px, py)
    return area(radius)

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

def area(radius):
    return  math.pi * radius**2
print calcArea(0, 0, 0, 1)
```

Calling distance and area from within calcArea

More Recursion- Fibonacci Example

- Fibonacci number
- Definition of the Fibonacci sequence

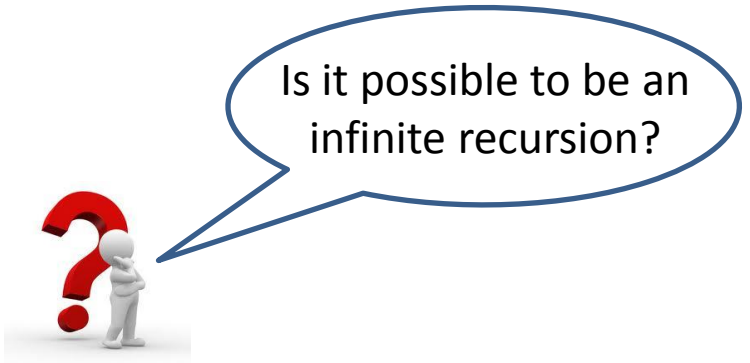
```
fibonacci(0) = 1                                          # base case
fibonacci(1) = 1                                          # base case
fibonacci(n) = fibonacci(n – 1) + fibonacci(n – 2); # general case
```

Here is a Fibonacci sequence:

1, 1, 2, 3, 5, 8, 13, …

```
"""
input:
    n: an integer value
note:
    This is a recursive function
    to evaluate a Fibonacci number.
    The input should be an integer
"""
def fibonacci (n):
    if n == 0 or n == 1: # base case
        return 1
    else:
        return fibonacci(n-1) + fibonacci(n-2) # general case

# test cases
print 'Fibonacci 0', fibonacci(0)
print 'Fibonacci 1', fibonacci(1)
print 'Fibonacci 2', fibonacci(2)
print 'Fibonacci 3', fibonacci(3)
print 'Fibonacci 15', fibonacci(15)
```



Boolean Function

- A Boolean function is a function that returns a Boolean value True or False.

```
"""
Purpose: to check if x can be divided by y
Note:  x, y must be integer values
Parameters:
Input: two input parameters
     x: the numerator; y: the denominator
return value:
     True or False
"""
def isDivisible(x, y):
    if x % y == 0:
        return True
    else:
        return False

print "'10 can be divided by 3' is ", isDivisible(10, 3)
print "'10 can be divided by 5' is ", isDivisible(10, 5)
```

Checking Types

- A useful Boolean function

```
>>> isinstance(2, int)
True

>>> isinstance(2, str)
False

>>> isinstance('2', str)
True

>>> isinstance(2, float)
False

>>> isinstance(2.1, float)
True
```

Optimize isDivisible

```
"""
 Purpose: to check if x can be divided by y
 Note:   x, y must be integer values
"""
def isDivisible(x, y):
    if (not isinstance(x, int) ) or (not isinstance(y, int) ):
        print 'Input must be integers.'
        return False
    elif x % y == 0:
        return True
    else:
        return False

print '10 can be divided by 3.1 is: ', isDivisible(10, 3.1)
print '10 can be divided by 5 is: ', isDivisible(10, 5)
```

Optimize factorial

```
"""
input:
    n: an integer to calculate its factorial value
note:
    This is a recursive function
    to evaluate a factorial value.
    The input must be an integer
"""
def factorial(n):
    if (not isinstance(n, int)) or (n < 0):
        print 'Input must be an integer.'
        return -1
    elif n == 0:        # base case
        return 1
    else:               # general case
        return  n * factorial(n-1)

# two test cases
print factorial(3)
print factorial(8.2)
```

Optimize fibonacci

```
"""
input:
    n: an integer value
note:
    This is a recursive function
    to evaluate a Fibonacci number.
    The input must be an integer
"""
def fibonacci (n):
    if (not isinstance(n, int)) or (n < 0):
        print 'Input must be a positive integer.'
        return -1
    elif n == 0 or n == 1:
        return 1
    else:
        return fibonacci(n-1) + fibonacci(n-2)

# test cases
print fibonacci(10)
print fibonacci(5.2)
```

Summary

- A fruitful function returns a value (write a return in every possible path)
- Incremental development is recommended!
- Fibonacci Example
- Boolean functions
- Type check: isinstance(…, …)