Programação I

Licenciatura em Engenharia Informática

2015–2016

Vitor Beires Nogueira

Escola de Ciências e Tecnologia
Universidade de Évora

Funções com Retorno

- Já vimos algumas funções que retornam algo
- As funções que fizemos até agora não retornam nada, i.e. retornam `None`

Exemplo (função `area`)

```
def area(radius):
    temp = math.pi * radius**2
    return temp
```

Exemplo (função `area`: versão sem variável local)

```
def area(radius):
    return math.pi * radius**2
```

Múltiplos returns

Exemplo (função `absolute_value`)

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

Exemplo (Dead code)

```
def absolute_value(x):
    if x >= 0:
        return x
    else:
        return -x
    print('Hello')
```

No return?

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

Desenvolvimento Incremental I

Suponha que pretendemos calcular a distância (Euclidiana) entre dois pontos:

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

Exemplo (`distance` v0.1)

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

Exemplo (`distance` v0.2)

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

Desenvolvimento Incremental II

Exemplo (distance v0.3)

```
def distance(x1, y1, x2, y2):
    dx = x2 - x1
    dy = y2 - y1
    dsquared = dx**2 + dy**2
    print('dsquared is', dsquared)
    return 0.0
```

Exemplo (distance v0.4)

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

Desenvolvimento Incremental III

Exemplo (`distance` v1.0)

```
def distance(x1, y1, x2, y2):
    return math.sqrt((x2 - x1)**2 + (y2 - y1)**2)
```

Composição

Exemplo (função `circle_area`)

```
def circle_area(xc, yc, xp, yp):
    return area(distance(xc, yc, xp, yp))
```

Funções Booleanas I

Exemplo (função `is_divisible`)

```
def is_divisible(x, y):
    if x % y == 0:
        return True
    else:
        return False
```

Qual será a versão minimalista de `is_divisible`?

Exemplo (Mau uso de uma função booleana)

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

Funções Booleanas II

Exemplo (Uso adequado de uma função booleana)

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

Função factorial

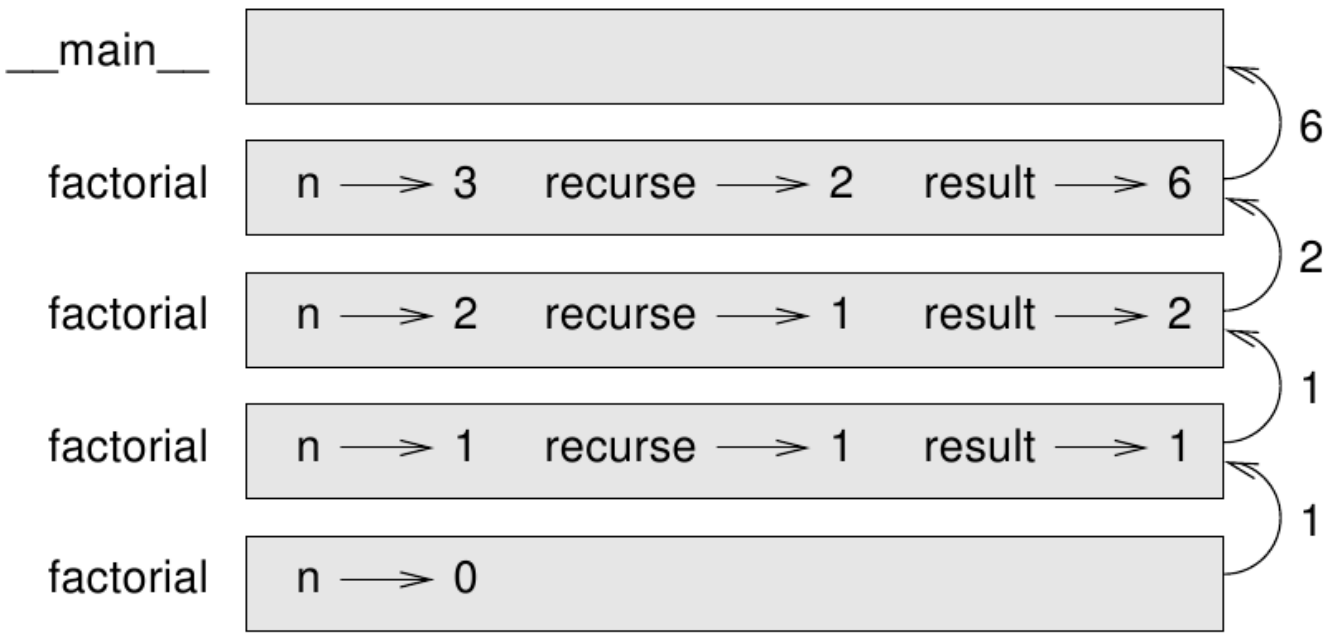
Definição (Factorial)

$$0! = 1$$
$$n! = n(n-1)!$$

Exemplo (função `factorial`)

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

Factorial: stack diagram



Factorial: minimalista

Será que conseguimos definir uma versão minimalista do factorial?

Exemplo (função `factorial`)

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

Definição (Fibonacci)

$$fibonnaci(0) = 0$$
$$fibonnaci(1) = 1$$
$$fibonnaci(n) = fibonnaci(n-1) + fibonnaci(n-2), n \geq 2$$

Exemplo (função `fibonnaci`)

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

Verificação de Tipos

O que acontece se tentarmos calcular o factorial de 1.5?

```
>>> factorial(1.5)
RuntimeError: Maximum recursion depth exceeded
```

Exemplo

```
def factorial (n):
  if not isinstance(n, int):
    print 'Factorial is only defined for integers.'
    return None
  elif n < 0:
    print 'Factorial is only defined for positive integers
    return None
  elif n == 0:
    return 1
  else:
    return n * factorial(n-1)
```