Recursividade

Programação I 2017.2018

Teresa Gonçalves tcg@uevora.pt

Departamento de Informática, ECT-UÉ

Como programar?



Perceber o problema

Dados e Resultados

Pensar numa solução

Dividir o problema em problemas mais simples

Resolver os problemas mais simples

Resolver o problema mais complexo

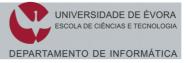
Implementar a solução

Utilizar funções para estruturar a resolução dos sub-problemas

Testar a solução

Fazer vários testes

Escolher valores que induzam comportamentos diferentes do programa



Sumário

Revisões Recursividade Argumentos



Funções (revisão)

Função

Sequência de instruções com nome que realiza uma computação

Uma função

Tem um nome

Recebe argumentos

Devolve um resultado

É executada sempre que o seu nome é invocado

Definição vs. Utilização



Definição de uma função

Especifica

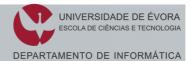
o nome e parâmetros da função a sequência de instruções a executar

Possui

Cabeçalho: nome, parâmetros

Corpo: instruções a executar

```
def print_disciplinas():
    print( 'Sistemas Digitais')
    print( 'Programação I')
```



Utilização (invocação)

Execução da sequência de instruções especificada na definição

Utilizando os argumentos indicados na invocação

Exemplo

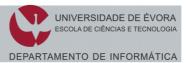
```
>>> print_disciplinas()
Sistemas Digitais
Programação I
```

Notas

A definição cria a função

As instruções só são executadas quando a função é invocada

Uma função tem de ser definida antes de ser invocada



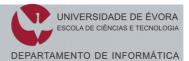
Argumentos e parâmetros

Argumento

Valor fornecido a uma função aquando da sua invocação

Parâmetro

Nome utilizado na função para referir o valor passado como argumento



Visibilidade de variáveis e parâmetros

Variáveis e parâmetros são locais

São apenas visíveis na função onde foram definidos

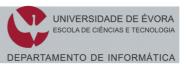
Fluxo de execução

A invocação de uma função provoca um desvio no fluxo de execução

Salta para o corpo da função

Executa as instruções lá existentes

Regressa, retomando o ponto onde tinha ficado





Recursividade

Corpo da função == sequência de instruções

Atribuições

Outras instruções

Invocação de funções

Recursividade

Quando no corpo da função contém a invocação à própria função

Exemplo - contagem decrescente

```
def ctg_decr(n):
    while n>=0:
        print(n)
        n=n-1
```

```
def ctg_decr_rec(n):
    print(n)
    if n>0:
        ctg_decr_rec(n-1)
```

ctg_decr_rec(3)

```
def ctg_decr_rec(n):
    print(n)
    if n>0:
        ctg_decr_rec(n-1)
```

```
ctg_decr_rec(3)
  print(3)
  ctg_decr_rec(2)
    print(2)
    ctg_decr_rec(1)
      print(1)
      ctg_dec_rec(0)
        print(0)
      (termina)
    (termina)
  (termina)
(termina)
```

Critério de paragem

É essencial

Caso contrário existem infinitas invocações sucessivas! Que esgota, os recursos de memória do computador

```
>>> def ctg_decr_rec(n):
    print(n)
    ctg_decr_rec(n-1)

>>> ctg_decr_rec(4)
... RuntimeError: maximum recursion depth exceeded
```

Factorial

```
n! = n*(n-1)*(n-2)*...*3*2*1
```

```
def factorial(n):
    res = 1
    while n>0:
        res = res * n
        n = n-1
    return res
```

Factorial

```
n! = n*(n-1)!
tem uma definição recursiva!!!

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

Fibonacci

```
fib(n)=fib(n-1)+fib(n-2)
 def fib(n, prof):
 # ver a profundidade
 barra=''
   i=1
   while i<=prof:
       barra=barra+'=='
       i = i+1
    print("%d %s fib(%d)"%(prof,barra,n))
   # a funcao
   if n<2:
      return 1
   else:
      f=fib(n-1,prof+1)+fib(n-2,prof+1)
      return f
```

fib(5,1)

```
1 == fib(5)
2 ==== fib (4)
3 = = = = = fib(3)
4 = = = = = = fib(2)
                  # devolve 1
5 ====== fib (1)
5 ====== fib (0)
                  # devolve 1
4 ====== fib(1) # devolve 1
3 = = = = = fib(2)
4 ====== fib(1) # devolve 1
4 = = = = = = fib(0) # devolve 1
2 = = = fib(3)
3 = = = = = fib(2)
4 = = = = = = = fib(0) # devolve 1
3 = = = = = fib(1) # devolve 1
```

Soluções recursivas

Cadeia de invocações da mesma função

Mas com parâmetros diferentes...

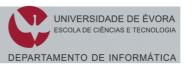
 $factorial(n) \rightarrow factorial(n-1) \rightarrow factorial(n-2) \rightarrow ...$

Critério de paragem

n>1

Trata parte do problema e "junta" com o resultado das restantes invocações

n*resultado_da_funcao





Utilização de valores por omissão

Valor por omissão

Valor utilizado quando a invocação não passa o argumento

```
def despedida(mensagem='Adeus!'):
   print(mensagem)

>>> despedida('Tchau!')
Tchau
>>> despedida()
Adeus!
```

Utilização de pares nome=valor

Par nome=valor

Identificação do parâmetro quando são utilizados valores por omissão

```
def repete(frase='teste', n=2):
    while n>0:
        print(frase)
        n=n-1
>>> repete()
>>> repete('outra frase')
>>> repete(n=4)
>>> repete(n=4, frase='outra frase')
```