Revisões

Programação I 2017.2018

Teresa Gonçalves tcg@uevora.pt

Departamento de Informática, ECT-UÉ

Sumário

Revisões
Como programar?
Exercício



Valores e variáveis

Valor

Elemento básico

Variável

Nome que representa um valor

Atribuição

Instrução que associa um valor à variável

$$x = a$$

Instruções, expressões e operações

Instrução

Unidade de código a ser executada

Expressão

Conjunto de operações sobre valores e variáveis

Operação

Cálculo indicado através de operadores e operandos



Operadores, operandos e precedência

Operador

Símbolo que representa um cálculo

Aritméticos: + - * / % // **

Relacionais: == != < <= > >=

Operando

Argumentos dos operadores

Podem ser valores ou variáveis

Precedência

Regras que definem a ordem de avaliação das expressões

$$() > ** > * / > + -$$



Tipos

Inteiro: int

Não tem limite máximo

Real: float

Melhor aproximação aos números reais

Complexos: complex

Cadeia de caracteres: str

Operações: concatenação, repetição

Booleano: boolean

Representa um valor verdade



Boolean

Valores

False, True

False: 0, sequência vazia

True: restantes

Operadores lógicos

and, or, not

Avaliação mínima ou "short circuit"

Precedência

not > and > or

Operadores relacionais

== != < <= > >=



Utilização

Comparações ou testes

Igualdade, desigualdade

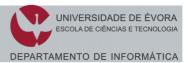
Expressar um conjunto de características

Através de conjunções (and), disjunções (or) e outras combinações

Exemplos

Verificar se x é potência 2 de y

Quais os números divisíveis por 5 e múltiplos de 3?



Conversão de tipos

Conversão implícita

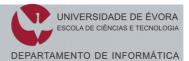
Conversão automática para o tipo mais abrangente

```
boolean → int
int → float
float → complex
```

Conversão explícita

Através de funções específicas

```
float()
int()
string()
eval()
```



Execução condicional

Instrução if

```
if <condição>:
     <instruções quando a condição é verdadeira>
```

Instrução if-else

```
if <condição>:
        <instruções quando a condição é verdadeira>
else:
        <instruções quando a condição é falsa>
```

Instrução if-elif

```
if <condição1>:
    <instruções quando condição1 é verdadeira>
elif <condição2>:
    <instruções quando condição1 é falsa e condição2</pre>
é verdadeira>
```

Atenção!

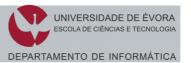
É obrigatório existir instruções no corpo do if

Pode não existir um else

Podem existir inúmeros elif

Apenas é executado um dos ramos

Apenas as instruções referentes à 1º condição verdadeira são executadas



Iteração

Instrução while

```
while <condicao>:
     <bloco de instruções>
```

Fluxo de execução

- 1. Avaliar a condição, obtendo True ou False
- 2.Se False, sai da instrução while e continua com próxima instrução
- 3.Se True, executa o corpo do while e volta ao passo 1

Instrução break

Termina o ciclo antecipadamente

Instrução continue

Passa de imediato para nova iteração

fazendo o teste e não executando as instruções restantes do bloco



Atenção!

Verificar o valor das variáveis da condição

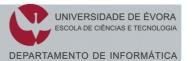
Para o ciclo terminar devem ser alteradas no corpo do while

Verificar a condição de paragem

É comum existir uma iteração a mais ou a menos

Verificar efeitos da interrupção antecipada

As instruções break e continue podem produzir efeitos não desejados no comportamento do ciclo



Função

Sequência de instruções com nome que realiza uma computação

Tem um nome

Recebe argumentos

Devolve um resultado

É executada sempre que o seu nome é invocado

Definição da função

Especificação do nome e parâmetros e sequência de instruções a executar

Invocação da função

Execução das instruções especificadas na definição da função



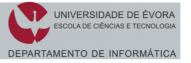
Fluxo de execução

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

Salta para o corpo da função

Executa as instruções lá existentes

Regressa, retomando o ponto onde tinha ficado



Argumentos, parâmetros e resultado

Argumento

Valor fornecido a uma função aquando da sua invocação

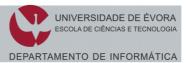
Parâmetro

Nome utilizado na função para referir o valor passado como argumento

Resultado

Valor devolvido pela função

Instrução return (com valor indicado à direita)



Argumentos

Podem ser

valores, variáveis, expressões, funções

Avaliação dos argumentos

Antes da invocação da função

$$f(g(x)) = f \circ g(x)$$

Visibilidade de variáveis e parâmetros

São locais

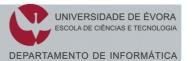
Apenas visíveis na função onde foram definidos

Então

Podem existir funções diferentes com variáveis com o mesmo nome

Uma variável local esconde outra com o mesmo nome

Dentro da função é usada a definição mais próxima do nome Fora da função é usada a definição visível



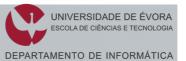
Porquê usar funções?

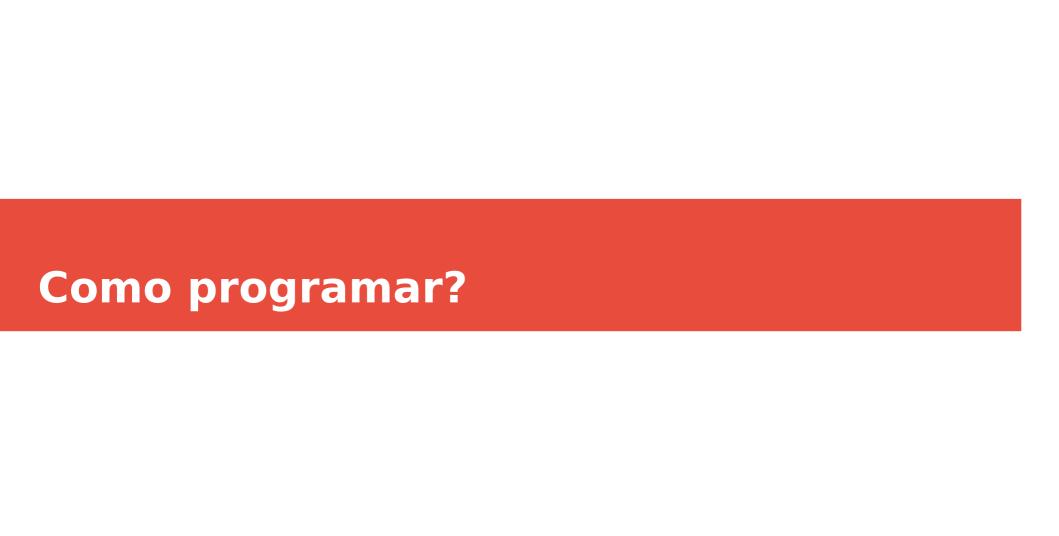
Torna o programa mais legível

Torna mais fácil fazer debugging

Permite analisar cada uma das partes em separado

Pode tornar o programa mais pequeno ao eliminar código repetido





Como programar?

Processo de desenvolvimento

- 1. Compreender o problema
- 2. Conceber o algoritmo
- 3. Implementar o algoritmo
- 4. Testar

Como aprender?

Estudar, estudar, ...

Praticar, praticar, ...

Cometer erros, cometer erros, ...

Aprender com os erros, ...



Encontrar os números primos até ao número n

1. Compreender o problema

Input

n (inteiro)

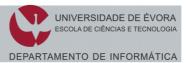
Resultado

mostrar os números primos entre 1 e n

Exemplo

Input: n=10

Resultado: 1, 3, 5, 7



2. Conceber o algoritmo

Para todos os números y até n, verificar se é primo

Para todos os números y até n

Verificar se é primo



3. Implementar o algoritmo

```
n = int(input('Introduza o limite: '))
i = 1
while i<=n:
    if primo(i):
        print(i)
    i = i+1</pre>
```

Indicar se um número é primo

1. Comprender o problema

Input

num (inteiro)

Resultado

Indicação se é primo (booleano)

Exemplo

Input: 4

Resultado: False



2. Conceber o algoritmo

Um número é primo se apenas for divisivel por si próprio...

Para todos os números y até num, verificar se num é divisivel por y

Para todos os números y até num

Verificar se num é divisivel por y

3. Implementar o algoritmo

```
Função primo(n)
def primo(num):
  y = ?
  while ?:
     if num%y==0: % num divisivel por i
        return False
     y = y+1
  return True
```

4. Testar

```
def primo(num):
    y = 2
    while y<num:</pre>
        if num%y==0: % num divisivel por y
            return False
        y = y+1
    return True
n = int(input('Introduza o limite: '))
i = 1
while i<=n:
    if primo(i):
        print(i)
    i = i+1
```

Ciclo while

É necessário verificar se num é divisivel por y para todos os valores entre 2 e num?

