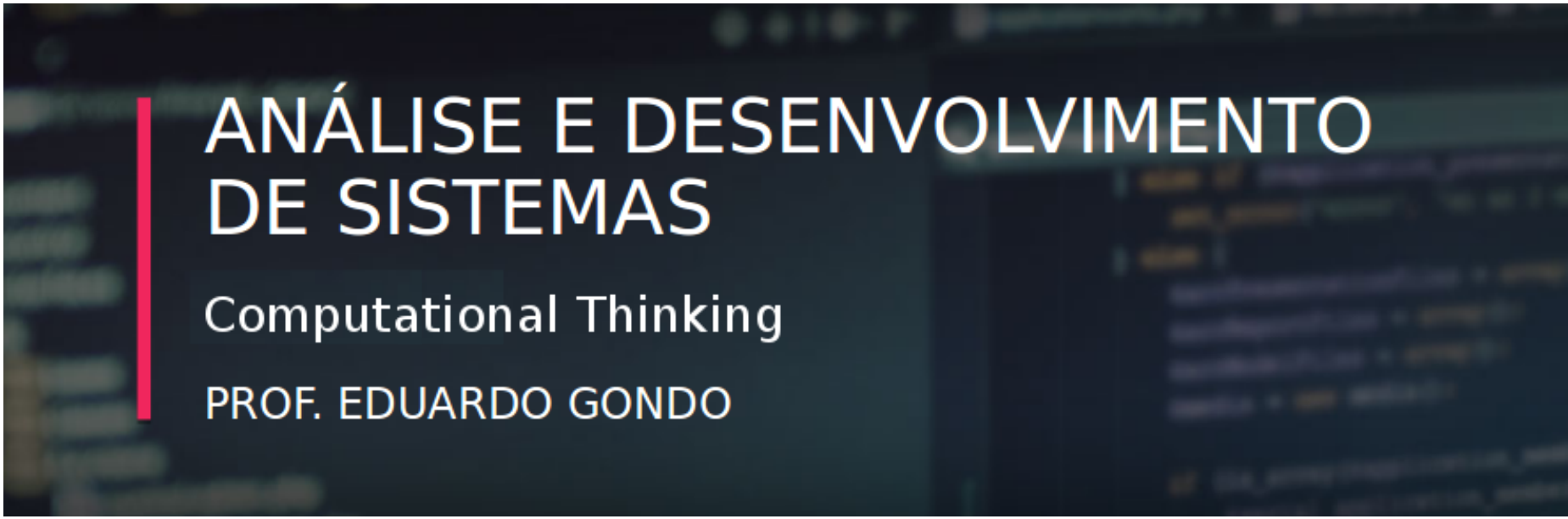
FIAP GRADUAÇÃO


ANÁLISE E DESENVOLVIMENTO DE SISTEMAS
Computational Thinking
PROF. EDUARDO GONDO

Agenda

- necessidade
- while
- exercícios

necessidade

PROBLEMA 4.1: Dados uma sequência de 5 números inteiros. Calcule a soma de todos os números da sequência.

Vamos resolver esse nosso primeiro problema usando a linguagem Python utilizando apenas duas variáveis.

```
1   soma = 0
2
3   aux = input("Digite um número: ")
4   num = int(aux)
5   soma = soma + num
6
7   aux = input("Digite um número: ")
8   num = int(aux)
9   soma = soma + num
10
11  aux = input("Digite um número: ")
12  num = int(aux)
13  soma = soma + num
```

Solução do Problema 1 em Python

```
14
15  aux = input("Digite um número: ")
16  num = int(aux)
17  soma = soma + num
18
19  aux = input("Digite um número: ")
20  num = int(aux)
21  soma = soma + num
22
23  println("O resultado é ", soma)
```

necessidade

PROBLEMA 4.2: Dado um número inteiro positivo n, escreva um algoritmo que imprime a tabuada de n até o valor 10.
SOLUÇÃO: Supondo que o valor digitado seja $n = 6$, seu programa deverá imprimir:

```
6 x 1 = 6
6 x 2 = 12
6 x 3 = 18
6 x 4 = 24
6 x 5 = 30
6 x 6 = 36
6 x 7 = 42
6 x 8 = 48
6 x 9 = 54
6 x 10 = 60
```

Solução do Problema 2

```
1   aux = input("Digite um numero: ")
2   n = int(aux)
3
4   print(n, "x 1 =", (n * 1))
5   print(n, "x 2 =", (n * 2))
6   print(n, "x 3 =", (n * 3))
7   print(n, "x 4 =", (n * 4))
8   print(n, "x 5 =", (n * 5))
9   print(n, "x 6 =", (n * 6))
10  print(n, "x 7 =", (n * 7))
11  print(n, "x 8 =", (n * 8))
12  print(n, "x 9 =", (n * 9))
13  print(n, "x 10 =", (n * 10))
```

Considerações sobre a tabuada

- na tabuada as instruções não são exatamente iguais como no problema anterior
- mas é fácil modificar o algoritmo para deixá-lo exatamente com as mesmas instruções
- para isso, basta inserirmos uma variável *i* inteira que irá assumir os valores de 1 a 10
- entre cada instrução de impressão, devemos colocar uma instrução que aumentará a variável *i* em uma unidade
- veja como fica a Tabuada com as modificações

Solução do Problema 2

Início:

Solução do Problema 2

```
1   aux = input("Digite um numero: ")
2
3   n = int(aux)
4   i = 1
5   print("{} x {} = {}".format(n, i, (n*i)))
6   i = i + 1
7   print("{} x {} = {}".format(n, i, (n*i)))
8   i = i + 1
9   print("{} x {} = {}".format(n, i, (n*i)))
10  i = i + 1
11  print("{} x {} = {}".format(n, i, (n*i)))
12  i = i + 1
```

Comandos de repetição

- para os 2 problemas anteriores usamos instruções repetidas
- conseguimos resolver pois a quantidade de comandos repetidos é pequena
- porém para alguns problemas é necessário outra estrutura para nossos algoritmos: os comandos de repetição
- veja o problema abaixo:

PROBLEMA 4.3: Escreva um algoritmo que dados um número inteiro positivo n, imprime na tela todos os números de 1 a n.

Considerações

- é impossível para este problema adotar a mesma estratégia de solução
- note que o número n é uma informação fornecida pelo usuário e
- a quantidade de instruções depende diretamente do valor de n

Solução do Problema 3

Início:

Solução do Problema 3

```
1   aux = input("Digite a qtd de numeros: ")
2
3   n = int(aux)
4
5   num = 1
6   while num <= n:
7       print(num)
8       num = num + 1
9
10  print("Fim")
```

Comando de repetição while

Importante!

Os comandos de repetição são elementos fundamentais dentro de linguagens de programação pois permite a execução de um mesmo conjunto de instruções até que a condição não seja mais satisfeita.

Uma execução desse conjunto de instruções é chamado de iteração.
Na linguagem Python, temos o comando de repetição while:

- a sintaxe do comando **while** pode ser apresentada como:

```
1     while <expressão condicional>:
2         //bloco contendo instruções que
3         //serão executadas repetidamente
```

- só para relembrar, uma `<expressão condicional>` é uma expressão que retorna verdadeiro (`True`) ou falso (`False`)

while — continuação

- interpretação: **enquanto a expressão condicional for verdadeira, as instruções do bloco são executadas**
- testa a expressão condicional antes de entrar no bloco, caso a expressão seja falsa, nenhuma instrução do bloco é executada
- ao término da execução das instruções do bloco, testa-se novamente a expressão condicional
- se a expressão for falsa a repetição termina
- podemos dizer que o while executa as instruções do bloco 0 ou mais vezes
- na maioria dos algoritmos, alguma hora a expressão condicional deve se tornar falsa para que o comando de repetição não entre em execução infinita

Contador

- muitos problemas usam um contador associado ao comando while
- esses contadores são utilizados para controlar o número de iterações realizadas
- veja abaixo um exemplo:

```
1  contador = 1
2
3  while contador < limite:
4
5      #coloque aqui os comandos que
6      #serão repetidos
7      contador = contador + 1
8
9  print("Fim do while")
```

Problema da Validação

Em muitos dos algoritmos devemos fazer validações dos dados de entrada. Com os comandos de repetição temos condições de garantir que a informação esteja correta antes do algoritmo prosseguir. Vejamos o seguinte problema:

Problema 4.4: Escreva um programa que dadas duas notas de 0 a 10 calcula a média aritmética entre elas.

Solução do Problema 4

Solução:

Solução do Problema 4

```
1  aux = input("Digite a 1ª nota: ")
2  nota1 = float(aux)
3
4  while nota1 < 0 or nota1 > 10:
5      aux = input("Nota inválida, digite a 1ª nota: ")
6      nota1 = float(aux)
7
8  aux = input("Digite a 2ª nota: ")
9  nota2 = float(aux)
10
11 while nota1 < 0 or nota1 > 10:
12     aux = input("Nota inválida, digite a 2ª nota: ")
13     nota2 = float(aux)
14
15 media = (nota1 + nota2) / 2
16
17 print("A média vale {:.2f}".format(media))
```

Problema 4.5

Escreva um programa que dado um inteiro n positivo calcula e imprime a soma de todos os números inteiros entre 1 e n.

Por exemplo, se $n = 10$ então deverá ser calculado:

$$1 + 2 + 3 + 4 + 5 + 6 + 7 + 8 + 9 + 10 = 55$$

Solução do Problema 5

Solução:

Solução do Problema 4.5

```
1  aux = input("Digite um inteiro positivo: ")
2  n = int(aux)
3
4  while n < 0:
5      aux = input("Erro! Digite um inteiro positivo: ")
6      n = int(aux)
7
8  soma = 0;
9  i = 1
10 while i <= n:
11     soma = soma + i
12     i = i + 1
13
14 print("Valor da soma vale ", soma)
```

Exemplos

PROBLEMA 4.6 Escreva um algoritmo que recebe um inteiro positivo *num* e imprime todos os divisores positivos de *num*.

EXEMPLO: Suponha que $num = 28$, nessa situação devemos imprimir os números 1, 2, 4, 7, 14 e 28, que são todos os divisores do 28.

Exemplos

PROBLEMA 4.6 Escreva um algoritmo que recebe um inteiro positivo *num* e imprime todos os divisores positivos de *num*.

EXEMPLO: Suponha que $num = 28$, nessa situação devemos imprimir os números 1, 2, 4, 7, 14 e 28, que são todos os divisores do 28.

```
1  num = input("Digite um inteiro positivo: ")
2  divisor = 1
3  while divisor <= num:
4      if num % divisor == 0:
5          print(divisor)
6      divisor = divisor + 1
```

Referência Bibliográfica

- Puga e Rissetti - Lógica de Programação e Estrutura de Dados
- Ascêncio e Campos - Fundamentos da Programação de Computadores
- Forbelone e Eberspacher - Lógica de programação: a construção de algoritmos e estruturas de dados
- Documentação do Python - https://docs.python.org/3.8/
- Python Programming For Beginners: Learn The Basics Of Python Programming (Python Crash Course, Programming for Dummies) (English Edition). Kindle
- Python: 3 Manuscripts in 1 book: - Python Programming For Beginners - Python Programming For Intermediates - Python Programming for Advanced (English Edition). Kindle

Copyleft