

BCM0505-22 – Processamento da Informação

Laços - Parte 3

Carla Negri Lintzmayer
carla.negri@ufabc.edu.br
<http://professor.ufabc.edu.br/~carla.negri/>

Outline

Laços Aninhados

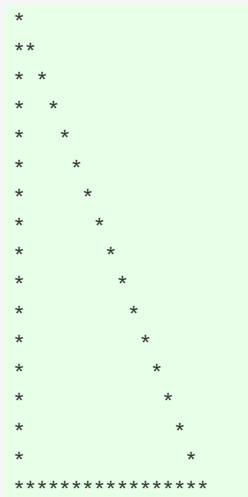
Break

Continue

Exercícios

Laços Aninhados

ASCII ART



Vamos desenhar um triângulo retângulo com ASCII Art

- **ASCII Art:** A ASCII art é uma forma de expressão artística usando apenas 95 caracteres imprimíveis (de um total de 128) definidos pelo padrão ASCII de 1963 (ASCII, abreviação do inglês para *American Standard Code for Information Interchange*).

ASCII ART

```
*
**
* *
*  *
*   *
*    *
*     *
*      *
*       *
*        *
*         *
*          *
*           *
*            *
*             *
*              *
*               *
*                *
*                 *
*                  *
*                   *
*                    *
*                     *
*                      *
*                       *
*                        *
*                         *
*                          *
*                           *
*                            *
*                             *
*                              *
*                               *
*                                *
*                                 *
*                                  *
*                                   *
*                                    *
*                                     *
*                                      *
*                                       *
*                                        *
*                                         *
*                                          *
*                                           *
*                                            *
*                                             *
*                                              *
*                                               *
*                                                *
*                                                 *
*                                                  *
*                                                   *
*                                                    *
*                                                     *
*                                                      *
*                                                       *
*                                                        *
*                                                         *
*                                                          *
*                                                           *
*                                                            *
*                                                             *
*                                                              *
*                                                               *
*                                                                *
*                                                                 *
*                                                                  *
*                                                                   *
*                                                                    *
*                                                                     *
*                                                                      *
*                                                                       *
*                                                                        *
*                                                                         *
*                                                                          *
*                                                                           *
*                                                                            *
*                                                                             *
*                                                                              *
*                                                                               *
*                                                                                *
*                                                                                 *
*                                                                                  *
*                                                                                   *
*                                                                                    *
*                                                                                     *
*                                                                                      *
*                                                                                       *
*                                                                                        *
*                                                                                         *
*                                                                                          *
*                                                                                           *
*                                                                                            *
*                                                                                             *
*                                                                                              *
*                                                                                               *
*                                                                                                *
*                                                                                                 *
*                                                                                                  *
*                                                                                                   *
*                                                                                                    *
*                                                                                                     *
*                                                                                                      *
*                                                                                                       *
*                                                                                                        *
*                                                                                                         *
*                                                                                                          *
*                                                                                                           *
*                                                                                                            *
*                                                                                                             *
*                                                                                                              *
*                                                                                                               *
*                                                                                                                *
*                                                                                                                 *
*                                                                                                                  *
*                                                                                                                   *
*                                                                                                                    *
*                                                                                                                     *
*                                                                                                                      *
*                                                                                                                       *
*                                                                                                                        *
*                                                                                                                         *
*                                                                                                                          *
*                                                                                                                           *
*                                                                                                                            *
*                                                                                                                             *
*                                                                                                                              *
*                                                                                                                               *
*                                                                                                                                *
*                                                                                                             *
```

Vamos desenhar um triângulo retângulo com ASCII Art

- **ASCII Art:** A ASCII art é uma forma de expressão artística usando apenas 95 caracteres imprimíveis (de um total de 128) definidos pelo padrão ASCII de 1963 (ASCII, abreviação do inglês para *American Standard Code for Information Interchange*).

<code>_ {v} _</code>	<code>(v)</code>	<code>('></code>	<code>()</code>
<code>/-\</code>	<code>//-\</code>	<code>/\</code>	<code>// \</code>
<code>(_/)</code>	<code>(_/)</code>	<code><(_</code>	<code>(\=/)</code>
<code>^ ^</code>	<code>^ ^</code>	<code>~~</code>	<code>~ ~</code>

ASCII Art de Triângulo em Python

```
1 altura = int(input())
2
3 for lin in range(1, altura + 1):
4     for i in range(1, lin + 1):
5         if i == 1 or i == lin or lin == altura:
6             print("*", end="")
7         else:
8             print(" ", end="")
9     print()
```

Lembre-se: o parâmetro `end=""` instrui o comando `print` a não quebrar a linha

ASCII Art de Triângulo em Python

```
1  altura = int(input())
2
3  print("*") #primeira linha
4
5  for lin in range(2, altura):
6      print("*", end="")
7      for i in range(2, lin):
8          print(" ", end="")
9      print("*")
10
11 #última linha
12 for i in range(altura):
13     print("*", end="")
14 print()
```

Listagem de Números Primos

Dado um número n , liste os n primeiros números primos.

```
1  from math import sqrt
2
3  qtd_primos_para_imprimir = int(input())
4
5  candidato_primo = 2
6  qtd_primos_impessos = 0
7  while qtd_primos_impessos < qtd_primos_para_imprimir:
8      divisor = 2
9      encontrei_divisor = False
10     while not encontrei_divisor and divisor <= sqrt(candidato_primo):
11         if candidato_primo % divisor == 0:
12             encontrei_divisor = True
13         divisor += 1
14     if not encontrei_divisor:
15         print(candidato_primo)
16         qtd_primos_impessos += 1
17     candidato_primo += 1
```

Listagem de Números Primos

- Podemos melhorar o código anterior: sabemos que 2 é o único número primo par. Assim, podemos modificar a forma de atualização da variável `candidato_primo` para que essa só assuma valores ímpares (isso reduz pela metade o número de iterações do laço externo).
- A mesma melhoria pode ser aplicada a `divisor`

Exercício: Faça essas melhorias!

Listagem de Números Primos - Versão 2

Dado um número n , lista os n primeiros números primos.

```
1  from math import sqrt
2
3  def eh_primo(num):
4      if num == 0 or num == 1:
5          return False
6      divisor = 2
7      encontrou_divisor = False
8      while not encontrou_divisor and divisor <= sqrt(num):
9          if num % divisor == 0:
10             encontrou_divisor = True
11             divisor += 1
12     return not encontrou_divisor
```

Listagem de Números Primos - Versão 2

```
1  qtd_primos_para_imprimir = int(input())
2
3  candidato_primo = 2
4  qtd_primos_impessos = 0
5  while qtd_primos_impessos < qtd_primos_para_imprimir:
6      if eh_primo(candidato_primo):
7          print(candidato_primo)
8          qtd_primos_impessos += 1
9      candidato_primo += 1
```

- Note como podemos esconder a complexidade dos laços aninhados usando funções

Equações Diofantinas

Uma *Equação Diofantina* é uma equação, tipicamente polinomial, de duas ou mais variáveis e de coeficientes inteiros, na qual apenas as soluções inteiras são de interesse.

$$37x + 23y = 101$$

Equações Diofantinas

Uma *Equação Diofantina* é uma equação, tipicamente polinomial, de duas ou mais variáveis e de coeficientes inteiros, na qual apenas as soluções inteiras são de interesse.

$$37x + 23y = 101$$

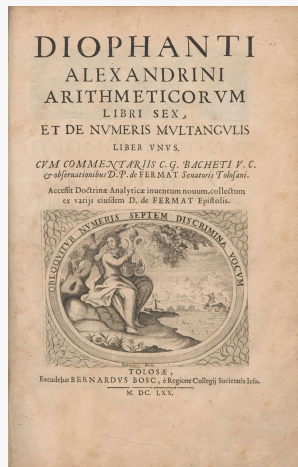
Soluções:

x	y
-47	80
-24	43
-1	6
22	-31
45	-68

Equações Diofantinas

Quando lidamos com (sistemas de) equações Diofantinas, temos mais incógnitas do que equações e estamos interessados em encontrar inteiros que resolvem simultaneamente todas as equações.

O nome *Diofantina* se refere ao matemático grego do século III Diofanto de Alexandria, considerado o pai da Álgebra.



Equações Diofantinas Lineares em Python

Faça um programa que lista as soluções para a seguinte equação diofantina linear:

$$ax + by + cz = 0$$

Equações Diofantinas Lineares em Python

Faça um programa que lista as soluções para a seguinte equação diofantina linear:

$$ax + by + cz = 0$$

```
1  a = int(input())
2  b = int(input())
3  c = int(input())
4
5  inf = int(input())
6  sup = int(input())
7
8  for x in range(inf, sup + 1):
9      for y in range(inf, sup + 1):
10         for z in range(inf, sup + 1):
11             valor = a*x + b*y + c*z
12             if valor == 0:
13                 print(f"solução: x = {x}, y = {y}, z = {z}")
```

Equações Diofantinas Lineares em Python

Pequena melhoria:

$$ax + by + cz = 0 \quad \Rightarrow \quad ax + by = -cz \quad \Rightarrow \quad \frac{ax + by}{-c} = z$$

Equações Diofantinas Lineares em Python

Pequena melhoria:

$$ax + by + cz = 0 \quad \Rightarrow \quad ax + by = -cz \quad \Rightarrow \quad \frac{ax + by}{-c} = z$$

```
1  a = int(input())
2  b = int(input())
3  c = int(input())
4
5  inf = int(input())
6  sup = int(input())
7
8  for x in range(inf, sup + 1):
9      for y in range(inf, sup + 1):
10         if (a*x + b*y) % (-c) == 0: # divisão dá número inteiro
11             z = (a*x + b*y) // (-c)
12             print(f"solução: x = {x}, y = {y}, z = {z}")
```

Break

Break

O comando **break** serve para finalizar imediatamente o laço mais aninhado

- Funciona com **while** e **for**



Break

```
1  for i in range(1, 8):  
2      if i == 4:  
3          break  
4      print(i)
```

Break

```
1  for i in range(1, 8):  
2      if i == 4:  
3          break  
4      print(i)
```

```
1  for i in range(1, 7):  
2      for j in range(i+1, 8):  
3          if j == 4:  
4              break  
5          print(i, j)
```

Como calcular a raiz quadrada de um número

Dado um x , queremos computar $y = \sqrt{x}$

Como calcular a raiz quadrada de um número

Dado um x , queremos computar $y = \sqrt{x}$

Método Babilônico (ou de Heron)

1. Seja y_1 uma estimativa para $y = \sqrt{x}$
 - Por exemplo, $y_1 = x$
 - Quanto melhor a estimativa mais rapidamente o algoritmo converge para a solução
2. Faça $y_n = \frac{1}{2}(y_{n-1} + \frac{x}{y_{n-1}})$
3. Se $|y_n - y_{n-1}|$ for “grande”, volte para 2
4. Devolva y_n

Método Babilônico em Python

```
1  ERRO = 1e-12
2
3  def mysqrt(num):
4      est_atual = num
5      while True:
6          est_anterior = est_atual
7          est_atual = (est_atual + num/est_atual) / 2
8          if abs(est_anterior - est_atual) <= ERRO:
9              break
10     return est_atual
11
12 valor = float(input())
13 raiz = mysqrt(valor)
14 print(f"raiz de {valor:.4f}: {raiz:.4f}")
```

Método Babilônico em Python

O **break** não é um comando necessário: podemos reescrever o código para eliminá-lo, mas ele é um comando conveniente que geralmente resulta em um código mais simples.

```
1  ERRO = 1e-12
2
3  def mysqrt(num):
4      est_atual = num
5      erro_pequeno = False
6      while not erro_pequeno:
7          est_anterior = est_atual
8          est_atual = (est_atual + num/est_atual) / 2
9          if abs(est_anterior - est_atual) <= ERRO:
10             erro_pequeno = True
11      return est_atual
12
13 valor = float(input())
14 raiz = mysqrt(valor)
15 print(f"raiz de {valor:.4f}: {raiz:.4f}")
```

Método Babilônico em Python

Outra possibilidade, sem `while` e sem variável controladora.

```
1  ERRO = 1e-12
2
3  def mysqrt(num):
4      est_anterior = num + ERRO
5      est_atual = num
6      while abs(est_anterior - est_atual) > ERRO:
7          est_anterior = est_atual
8          est_atual = (est_atual + num/est_atual) / 2
9      return est_atual
10
11 valor = float(input())
12 raiz = mysqrt(valor)
13 print(f"raiz de {valor:.4f}: {raiz:.4f}")
```

Continue

Continue

O comando `continue` serve para ignorar o restante do corpo do laço (while ou for) na iteração atual



Continue

```
1  for i in range(1, 8):  
2      if i == 4:  
3          continue  
4      print(i)
```

Continue

```
1  for i in range(1, 8):  
2      if i == 4:  
3          continue  
4      print(i)
```

```
1  for i in range(1, 7):  
2      for j in range(i+1, 8):  
3          if j == 4:  
4              continue  
5          print(i, j)
```

Um velho conhecido

Dado um n , calcule a soma dos n primeiros números ímpares.

```
1  n = int(input())
2  numero_atual = 0
3  soma = 0
4  qtd_impares_somados = 0
5  while qtd_impares_somados < n:
6      numero_atual += 1
7      if numero_atual % 2 == 0:
8          continue
9      soma += numero_atual
10     qtd_impares_somados += 1
11 print(f"soma dos {n} primeiros ímpares: {soma}")
```

Exercícios

Cada um com o seu quadrado

Dado um inteiro n , escreva um programa que imprima um quadrado de n asteriscos por n asteriscos. Abaixo é possível ver a saída esperada para o caso no qual $n = 5$.

```
*****  
*****  
*****  
*****  
*****
```

Tabuada

Escreva um programa que imprima a tabuada do 1 ao 10.

Sugestão de Saída:

```
1 x 1 = 1
1 x 2 = 2
...
2 x 1 = 2
2 x 2 = 4
...
10 x 1 = 1
10 x 2 = 20
...
10 x 10 = 100
```

Triângulo Numérico

Escreva um programa que, dado um número n , imprima um triângulo de n linhas onde a i -ésima linha do triângulo é composta pelos i primeiros naturais. Abaixo é possível ver o padrão de impressão qndo $n = 6$.

```
1
1 2
1 2 3
1 2 3 4
1 2 3 4 5
1 2 3 4 5 6
```

Triângulo de Pascal

O Triângulo de Pascal é um triângulo numérico infinito formado por números binomiais.

A n -ésima linha do triângulo contém n números, cada um sendo um coeficiente binomial da forma $\binom{n-1}{k}$, para k variando de 0 até $n - 1$:

	$k = 0$	$k = 1$	$k = 2$	$k = 3$	$k = 4$	$k = 5$	...
$n = 1$	$\binom{0}{0}$						
$n = 2$	$\binom{1}{0}$	$\binom{1}{1}$					
$n = 3$	$\binom{2}{0}$	$\binom{2}{1}$	$\binom{2}{2}$				
$n = 4$	$\binom{3}{0}$	$\binom{3}{1}$	$\binom{3}{2}$	$\binom{3}{3}$			
$n = 5$	$\binom{4}{0}$	$\binom{4}{1}$	$\binom{4}{2}$	$\binom{4}{3}$	$\binom{4}{4}$		
$n = 6$	$\binom{5}{0}$	$\binom{5}{1}$	$\binom{5}{2}$	$\binom{5}{3}$	$\binom{5}{4}$	$\binom{5}{5}$	
$\vdots$							

Lembre-se que $\binom{n}{k} = \frac{n!}{k!(n-k)!}$.

Faça um programa que imprima as n primeiras linhas do Triângulo de Pascal para um dado n .

Meus Primos

Escreva um programa que, dado um número inteiro n , imprima a decomposição de n em fatores primos.

Exemplo:

- Se $n = 17640$, então o seu programa deve imprimir:

$$17640 = 2 \times 2 \times 2 \times 3 \times 3 \times 5 \times 7 \times 7$$