Processamento da Informação

Estruturas de repetição - Parte 1

Prof. Jesús P. Mena-Chalco
CMCC/UFABC

Q1/2020

Estrutura de repetição: laço

Usado em situações em que é necessário repetir um determinado trecho de um programa, geralmente, um determinado número de vezes.

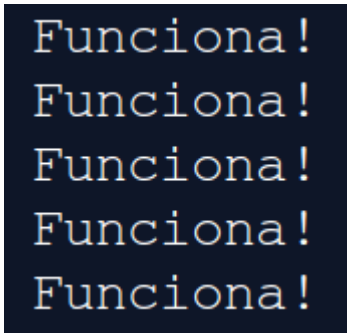
Duas formas:

- Escrever o trecho quantas vezes for necessário, ou
- Utilizar o conceito de **Laços**.

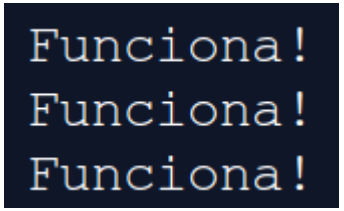
```
1    print("Funciona!")
2    print("Funciona!")
3    print("Funciona!")
4    print("Funciona!")
5    print("Funciona!")
6    print("Funciona!")
7    print("Funciona!")
8    print("Funciona!")
9    print("Funciona!")
10   print("Funciona!")
```

```
Funciona!
Funciona!
Funciona!
Funciona!
Funciona!
Funciona!
Funciona!
Funciona!
Funciona!
Funciona!
```

```
1   n = 1
2   while n<=10:
3       print("Funciona!")
```


Funciona!
Funciona!
Funciona!
Funciona!
Funciona!

...

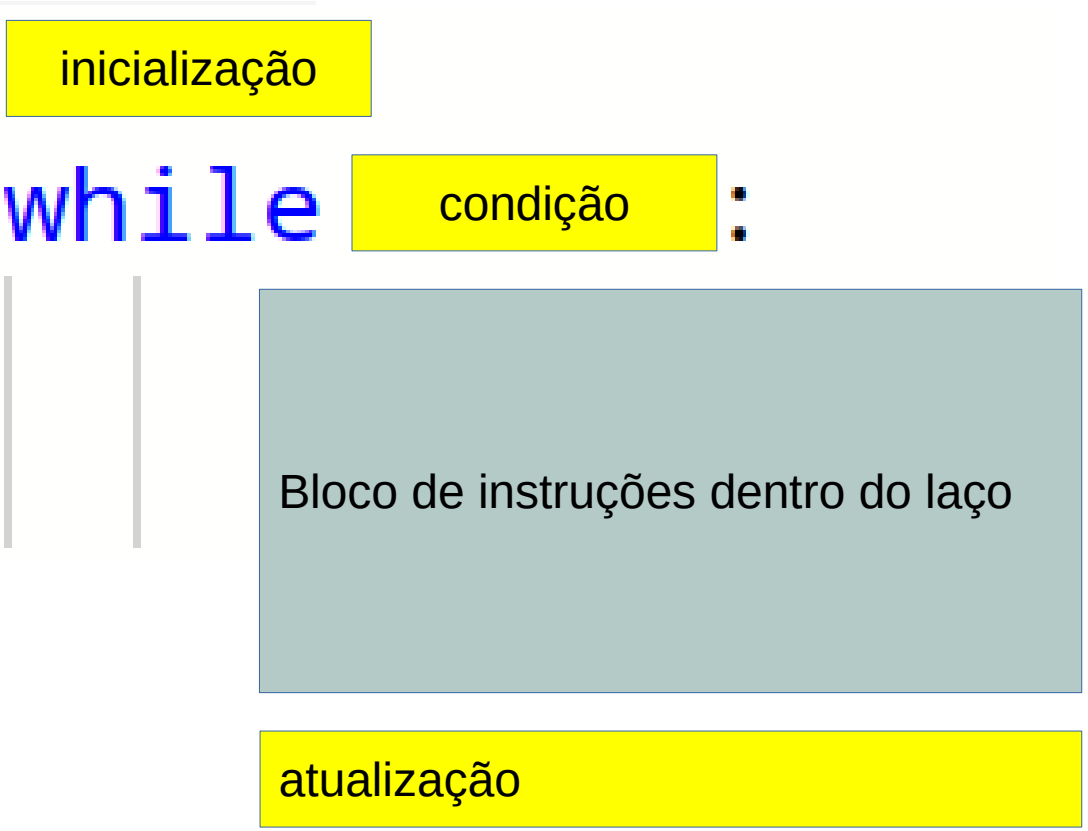

Funciona!
Funciona!
Funciona!

...

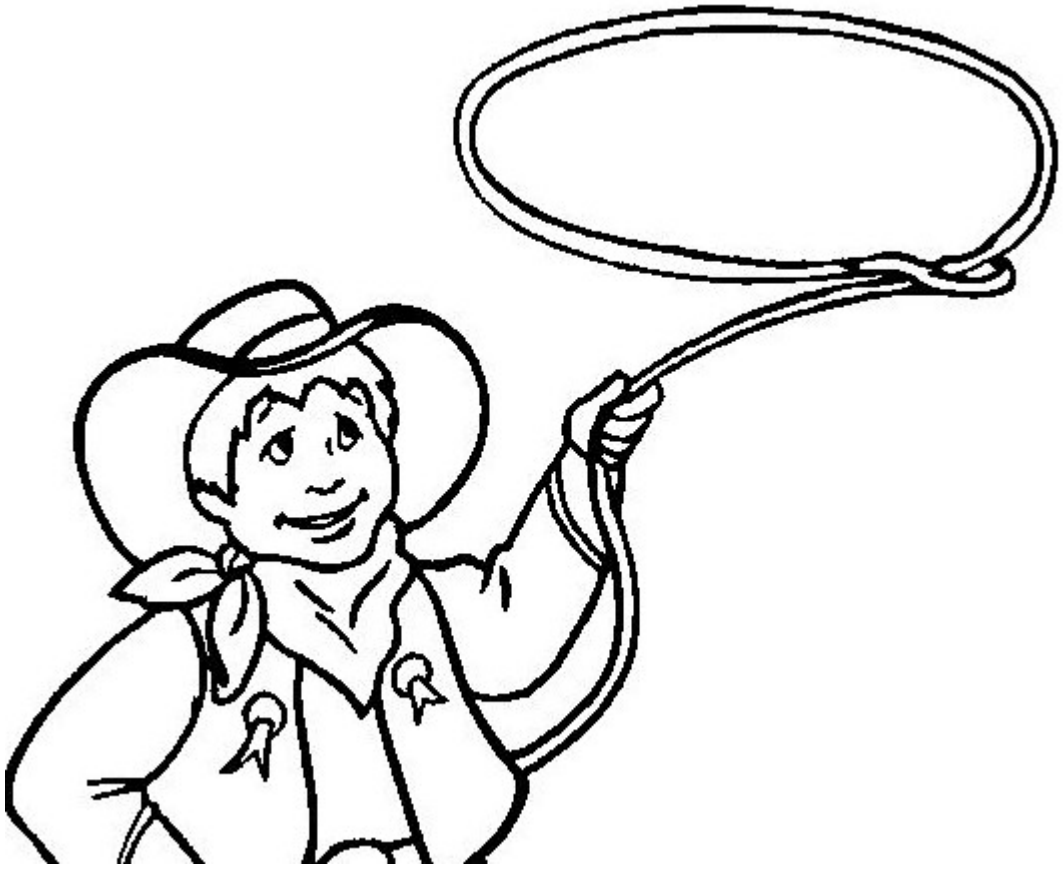
```python
n = 1
while n<=10:
    print("Funciona!")
    n = n+1
```

```
Funciona!
Funciona!
Funciona!
Funciona!
Funciona!
Funciona!
Funciona!
Funciona!
Funciona!
Funciona!
```

Estrutura de repetição: laço while



Estrutura de repetição: laço



Fonte: http://www.kumamon.co/

Exercício 01: Contagem regressiva

```
Faltam 5 segundos
Faltam 4 segundos
Faltam 3 segundos
Faltam 2 segundos
Faltam 1 segundos
Boom!
```

Exercício 01: Contagem regressiva

```
1   n = 5
2
3   while n>0:
4       print("Faltam {} segundos".format(n))
5       n = n-1
6
7   print("Boom!")
```

Exercício 02: Somatória de números inteiros

Dados dois inteiros, ***a*** e ***b***, com ***a<=b***, crie uma função que permita somar todos os números entre ***a*** e ***b***, eles incluídos.

```
def soma_intervalo(a: int, b: int) -> int:
```

Exercício 02: Somatória de números inteiros

```
def soma_intervalo(a: int, b: int) -> int:
    soma = 0
    n = a

    while n<=b:
        soma = soma + n
        n = n+1

    return soma
```

```
a = int(input())
b = int(input())
print(soma_intervalo(a,b))
```

Exercício 02: Somatória de números inteiros

```python
def soma_intervalo(a: int, b: int) -> int:
    soma = 0

    while a<=b:
        soma = soma + a
        a = a+1

    return soma
```

Exercício 02: Somatória de números inteiros

```python
def soma_intervalo(a: int, b: int) -> int:
    return (b**2 - a**2 + b + a)/2
```

Não é utilizado laço, mas é uma Solução elegante para a somatória

Exercício 03: SomaP

Crie um método, em que dado um inteiro **n>0**, permita somar a seguinte sequência:

$$1^2 + 2^2 + 3^2 + \cdots + n^2$$

Assinatura:

```
def somaP(n: int) -> int:
```

Exercício 03: SomaP

```
def somaP(n: int) -> int:
    soma = 0
    contador = 1

    while contador<=n:
        soma = soma + contador**2
        contador = contador + 1

    return soma
```

Exercício 03: SomaP

```
def somaP(n: int) -> int:
    soma = 0
    contador = 1

    while contador<=n:
        soma += contador**2
        contador += 1

    return soma
```

Exercício 04: SomaR

Crie uma função em que dado um inteiro n>0, seja realizada a seguinte somatória:

$$1 - 2 + 3 - 4 + 5 - 6 + \cdots + n$$

Assinatura: `def somaR(n: int) -> int:`

Exercício 04: SomaR

```
def somaR(n: int) -> int:
    soma = 0
    i = 1

    while i<=n:
        soma = soma + i*(-1)**(i+1)
        i = i + 1

    return soma
```

Exercício 05: Número triangular

Dizemos que um número natural é triangular se ele é produto de três número naturais consecutivos.
Dado um inteiro não negativo **n**, crie uma função que permita verificar se é triangular.

Exemplo:

- 120 é triangular, pois 4*5*6 = 120.
- 2730 é triangular, pois 13*14*15 = 2730.

Assinatura:

```
def numero_triangular(n: int) -> bool:
```

Devolve **True** se o número for triangular, caso contrário **False**

Exercício 05: Número triangular

```python
def numero_triangular(n: int) -> bool:
    i = 1

    while i*(i+1)*(i+2)<n:
        i = i + 1

    return i*(i+1)*(i+2) == n
```

```python
n = int(input())
print(numero_triangular(n))
```

Exercício 06: Aproximação de π

Uma aproximação de π bem conhecida é aquela calculada a partir da **Equação de Leibniz**:

$$1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \frac{1}{9} - \cdots = \frac{\pi}{4}$$

Crie uma função que permita calcular essa aproxiamção de π usando os primeiros **1000** termos.

Assinatura:

```
def pi() -> float:
```

Gottfried Wilhelm Leibniz

From Wikipedia, the free encyclopedia

"Leibniz" redirects here. For other uses, see Leibniz (disambiguation).

Gottfried Wilhelm (**von**) **Leibniz** (sometimes spelled **Leibnitz**) (/ˈlaɪbnɪts/;[11] German: [ˈgɔtfʁiːt ˈvɪlhɛlm fɔn ˈlaɪbnɪts][12][13] or [ˈlaɪpnɪts];[14] French: *Godefroi Guillaume Leibnitz*;[15] 1 July 1646 [O.S. 21 June] – 14 November 1716) was a prominent German polymath and one of the most important logicians, mathematicians and natural philosophers of the Enlightenment. As a representative of the seventeenth-century tradition of rationalism, Leibniz's most prominent accomplishment was conceiving the ideas of differential and integral calculus, independently of Isaac Newton's contemporaneous developments.[16] Mathematical works have consistently favored Leibniz's notation as the conventional expression of calculus. It was only in the 20th century that Leibniz's law of continuity and transcendental law of homogeneity found mathematical implementation (by means of non-standard analysis). He became one of the most prolific inventors in the field of mechanical calculators. While working on adding automatic multiplication and division to Pascal's calculator, he was the first to describe a pinwheel calculator in 1685[17] and invented the Leibniz wheel, used in the arithmometer, the first mass-produced mechanical calculator. He also refined the binary number system, which is the foundation of all digital computers.

In philosophy, Leibniz is most noted for his optimism, i.e. his conclusion that our universe is, in a restricted sense, the best possible one that God could have created, an idea that was often lampooned by others such as Voltaire. Leibniz, along with René Descartes and Baruch Spinoza, was one of the three great 17th-century advocates of rationalism. The work of Leibniz anticipated modern logic and analytic philosophy, but his philosophy also assimilates elements of the scholastic tradition, notably that conclusions are produced by

Gottfried Wilhelm Leibniz

Portrait by Christoph Bernhard Francke

Born	Gottfried Wilhelm von Leibniz 1 July 1646 Leipzig, Electorate of Saxony, Holy Roman Empire
Died	14 November 1716 (aged 70) Hanover, Electorate of Hanover, Holy Roman Empire

$$1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \frac{1}{9} - \cdots = \frac{\pi}{4}$$

$$\sum_{n=0}^{\infty} \frac{(-1)^n}{2n+1} = \frac{\pi}{4}$$

Exercício 06: Aproximação de π

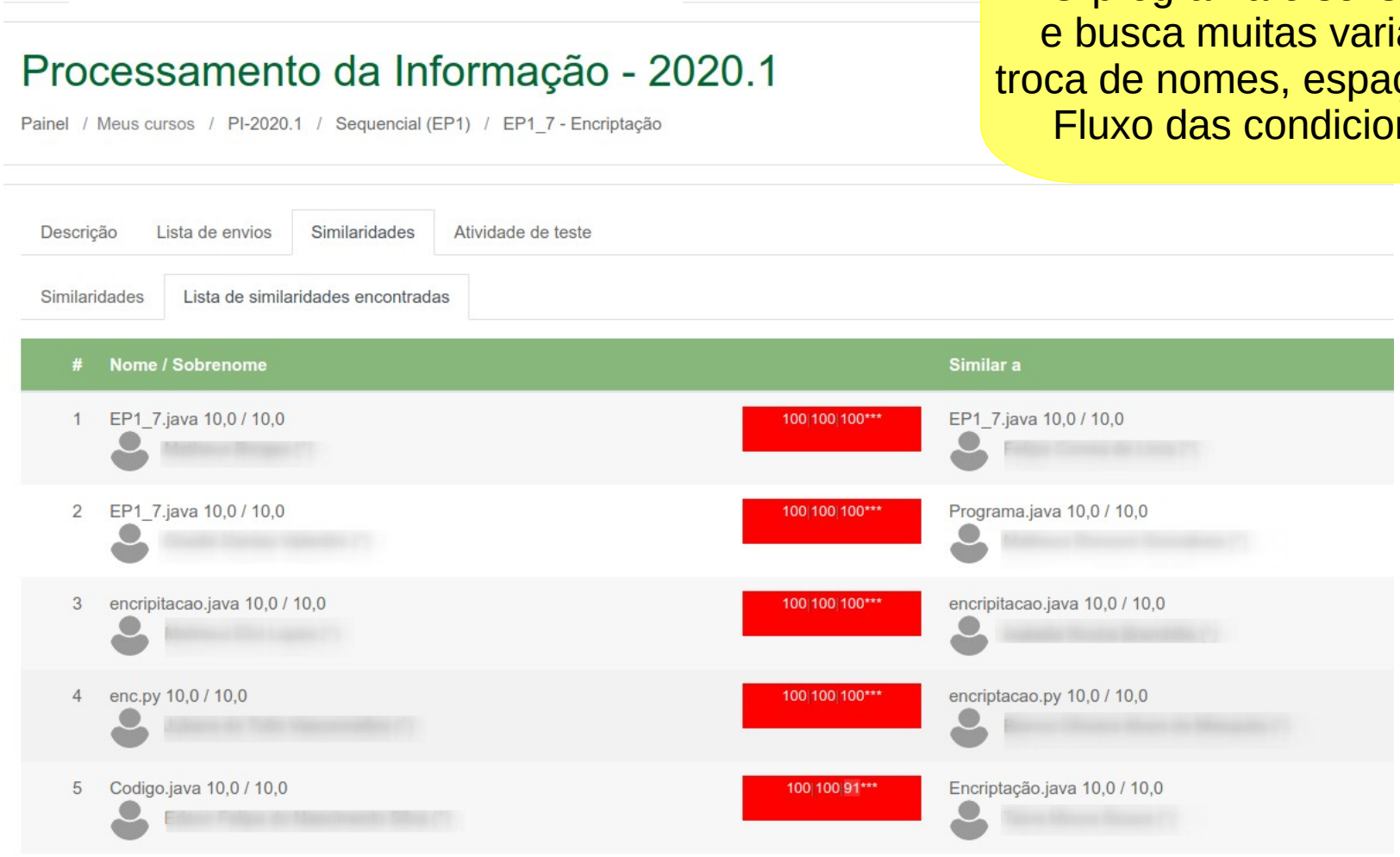
```
def pi() -> float:
    soma = 0
    i = 0

    while i<1000:
        soma = soma + (-1)**i/(2*i+1)
        i = i + 1

    return 4*soma
```

```
3.140592653839794
```

Atividade em aula

Cuidado com o ‘código amigo’!

O AVA permite identificar plágio em código fonte
O programa é sofisticado e busca muitas variações: troca de nomes, espaçamento, Fluxo das condicionais...



Processamento da Informação - 2020.1

Painel / Meus cursos / PI-2020.1 / Sequencial (EP1) / EP1_7 - Encriptação

Descrição | Lista de envios | Similaridades | Atividade de teste

Similaridades | Lista de similaridades encontradas

#	Nome / Sobrenome		Similar a
1	EP1_7.java 10,0 / 10,0	100 100 100***	EP1_7.java 10,0 / 10,0
2	EP1_7.java 10,0 / 10,0	100 100 100***	Programa.java 10,0 / 10,0
3	encripitacao.java 10,0 / 10,0	100 100 100***	encripitacao.java 10,0 / 10,0
4	enc.py 10,0 / 10,0	100 100 100***	encriptacao.py 10,0 / 10,0
5	Codigo.java 10,0 / 10,0	100 100 91***	Encriptação.java 10,0 / 10,0

https://ava.ufabc.edu.br/mod/vpl/similarity/diff.php?type1=1

95|99|94***

lista1_14.py

```
1  n = int(input())
2  c = input()
3  m = n
4  if c == "TROCA":
5
6
7
8
9
10
11
12
13
14
15
16
17 elif c == "INSERE":
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32 elif c == "APAGA":
33
34
35
36
37
38
39
40
41
42
43
44 print(m)
```

ex14.py

```
===  1  n = int(input())
===  2  c = input()
===  3  m = n
===  4  if c == "TROCA":
===  5
===  6
===  7
==#  8
===  9
==# 10
=== 11
==# 12
=== 13
==# 14
=== 15
==# 16
=## 17 elif c == "APAGA":
=== 18
<<<
=== 19
#"" 20
=== 21
### 22
=== 23
### 24
=== 25
<<<
<<
### 26
===  2
### 28
=## 29 elif c == "INSERE":
==# 30
>>> 31
==# 32
### 33
==# 34
### 35
==# 36
### 37
==# 38
>>> 39
>>> 40
### 41
==# 42
### 43
=== 44 print(m)
```

Questão 1

```
def m1(a: int) -> int:
    while (a>0):
        a = a-1
    return a
```

```
print(m1(10))
print(m1(0))
```

```
0
0
```

Questão 2

```
def m2(x: int, y: int) -> int:
    r = 0

    if x>y:
        t = x
        x = y
        y = t

    while x<=y:
        r = r+x
        x = x+1

    return r
```

Somatória dos inteiros no intervalo [x, y]

```
print(m2(10, 14))
print(m2(4, -3))
```

→

```
60
4
```

Questão 3

```
def m3(v: int) -> int:
    while (v>0 and v<10) or (v>10 and v<20):
        v = v+5

    return v
```

```
print(m3(10))
print(m3(1))
```

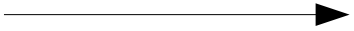
```
10
21
```

Questão 4

```
def m4(x: int) -> int:
    n = 1

    while x>1:
        n = n*x
        x = x-1

    return n
```

Fatorial de x

```
print(m4(3))
print(m4(6))
```



```
6
720
```

Questão 5

```
def m5(x: int) -> int:
    soma = 0

    p = 1
    while p<=x:
        s = 1
        while s<=x:
            soma = soma + 1
            s = s+1
        p = p+1

    return soma
```

x^2

```
print(m5(3))
print(m5(4))
```

→

```
9
16
```

Bônus - Desafio

```
def somatoria(n: int) -> int:
    soma = 0

    while n>=1:
        i = n
        while i>=1:
            soma = soma + n
            i = i-1

        n = n-1

    return soma
```

O método somatoria, dado um inteiro **n** positivo, calcula:

$$1^2 + 2^2 + 3^2 + \cdots + n^2$$

Teste de mesa

https://donkirkby.github.io/live-py-plugin/demo/

```python
def somatoria(n: int) -> int:
    soma = 0

    while n>=1:
        i = n
        while i>=1:
            soma = soma + n
            i = i-1

        n = n-1

    return soma


print(somatoria(4))
```

```
n = 4
soma = 0

                                               |                                    |                        |
i = 4                                          | i = 3                              | i = 2                  | i = 1
           |          |           |            |            |            |          |            |           |
soma = 4   | soma = 8 | soma = 12 | soma = 16  | soma = 19  | soma = 22  | soma = 25 | soma = 27  | soma = 29 | soma = 30
i = 3      | i = 2    | i = 1     | i = 0      | i = 2      | i = 1      | i = 0     | i = 1      | i = 0     | i = 0
                                               |                                    |                        |
n = 3                                          | n = 2                              | n = 1                  | n = 0

return 30


print('30')
```

Solução.py

Bateria de testes

AVA-UAFBC

A **Solução.py** é processada e avaliada com uma bateria de testes.

A comparação é apenas das saídas, isto é, da forma de impressão na tela.

Entrada	Saída do programa submetido para avaliação	Saída esperada
1	34	34
0.6	4.999	5.0