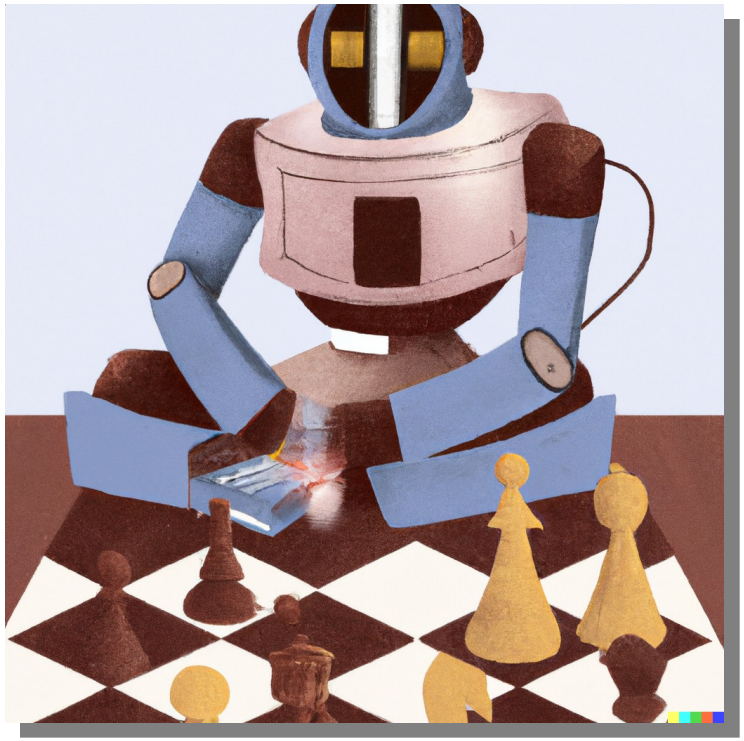
Universidade de Brasília
Departamento de Ciência da Computação



Semana 08

Iteração

CIC0004
Algoritmos e Programação de Computadores

Prof. Pedro Garcia Freitas https://pedrogarcia.gitlab.io/ pedro.garcia@unb.br

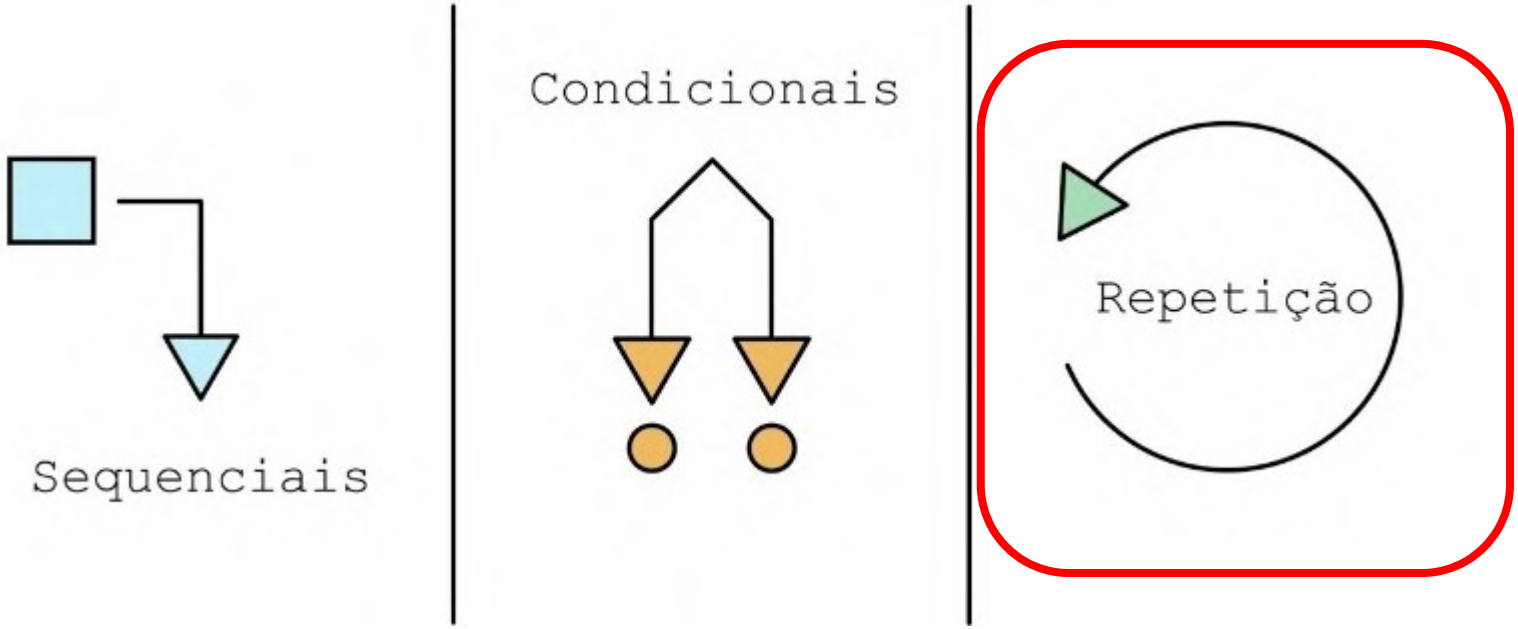
Brasilia

1. Objetivos

Esta aula revisita o conceito de repetição mostrado anteriormente e apresenta o conceito de iteração e como implementar tal conceito em linguagem Python.

2. Recapitulação

Nas aulas anteriores, vimos como estruturas algorítmicas as repetições com testes (no início e no final)



2. Recapitulação

Nas aulas anteriores, vimos como estruturas algorítmicas as repetições com testes (no início e no final)

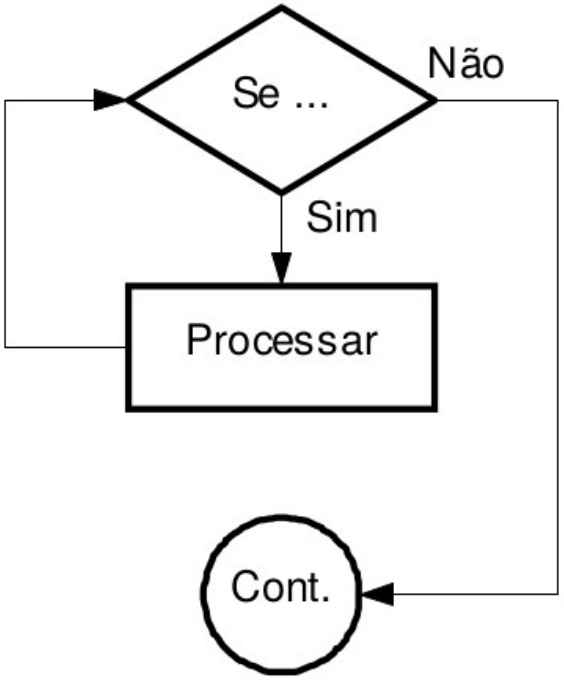
- **`enquanto`** ‹condição› **`faça`** ‹comandos›
- **`até que`** ‹condição› **`faça`** ‹comandos›
- **`faça`** ‹comandos› **`enquanto`** ‹condição›
- **`faça`** ‹comandos› **`até que`** ‹condição›

2. Recapitulação

Repetição com testes no início:

enquanto <condição> **faça** <comandos>

```
Algoritmo ExemploEnquanto
   ENQUANTO <condição> FAÇA
        processar
   FIM-ENQUANTO
Fim-Algoritmo
```



2. Recapitulação

Repetição com testes no início:

`enquanto` ‹condição› **`faça`** ‹comandos›

Exemplo 1: Escreva um algoritmo que solicita ao usuário um valor inteiro positivo, o lê, e imprime na tela do computador todos os número inteiros de 0 a N.

2. Recapitulação

```
ALGORITMO Mostra_N_Numeros
  VAR n
  VAR contador = 0
  IMPRIMIR("Digite o número N")
  LER(n)
  ENQUANTO contador <= N FAÇA
      IMPRIMIR(contador)
      Contador = contador + 1
  FIM-ENQUANTO
FIM-ALGORITMO
```

2. Recapitulação

```
ALGORITMO Mostra_N_Numeros
  VAR n
  VAR contador = 0
  IMPRIMIR(“Digite o número N”)
  LER(n)
  ENQUANTO contador <= N FAÇA
      IMPRIMIR(contador)
      Contador = contador + 1
  FIM-ENQUANTO
FIM-ALGORITMO
```

```
def main():
    print("digite o valor de N:")
    n = int(input())
    contador = 0
    while contador <= n:
        print(contador)
        contador += 1


if __name__ == "__main__":
    main()
```

2. Recapitulação

```
n = int(input("digite o valor de N:"))
```

```
ALGORITMO Mostra_N_Numeros
  VAR n
  VAR contador = 0
  IMPRIMIR("Digite o número N")
  LER(n)
  ENQUANTO contador <= N FAÇA
      IMPRIMIR(contador)
      Contador = contador + 1
  FIM-ENQUANTO
FIM-ALGORITMO
```

```
def main():
    print("digite o valor de N:")
    n = int(input())
    contador = 0
    while contador <= n:
        print(contador)
        contador += 1


if __name__ == "__main__":
    main()
```

2. Recapitulação

Shell

```
>>> %Run exercicio01.py
  digite o valor de N:12
  0
  1
  2
  3
  4
  5
  6
  7
  8
  9
  10
  11
  12

>>>
```

2. Recapitulação

Repetição com testes no início:

Exemplo 2: Escreva um programa que solicita ao usuário N valores reais positivos, calcula e imprime na tela do computador o somatório dos números digitados. O programa deve continuar solicitando valores até que o valor -1 seja digitado pelo usuário.

2. Recapitulação

```
def main():
    soma = 0
    n = 0
    while n != -1:
        n = int(input("Escreva um novo valor:"))
        soma += n
    print(f"O somatorio dos numeros digitados é {soma}")


if __name__ == "__main__":
    main()
```

2. Recapitulação

```
Shell
>>> %Run exercicio02.py
  Escreva um novo valor:12
  Escreva um novo valor:3
  Escreva um novo valor:5
  Escreva um novo valor:-1
  O somatorio dos numeros digitados é 19
>>>
```

2. Recapitulação

Repetição com testes no início:

Exemplo 3: O número 9801 possui a seguinte característica:

$$98 + 01 = 99$$

$$99^2 = 9801$$

Escreva um programa que encontre todos os números de quatro dígitos que apresentam tal propriedade.

2. Recapitulação

```
def main():
    inicio = 1000
    fim = 9999
    numero = inicio
    while numero <= fim:
        numero_as_str = str(numero)
        primeira_dezena = int(numero_as_str[0:2])
        segunda_dezena = int(numero_as_str[2:4])
        soma = primeira_dezena + segunda_dezena
        quadrado_da_soma = soma * soma
        if quadrado_da_soma == numero:
            out = (numero, primeira_dezena,
                   segunda_dezena, quadrado_da_soma)
            print("N=%d, N1=%d, N2=%d, soma2=%d" % out)
        numero += 1
```

2. Recapitulação

```
Shell
>>> %Run exercicio03.py
 N=2025, N1=20, N2=25, soma2=2025
 N=3025, N1=30, N2=25, soma2=3025
 N=9801, N1=98, N2=1, soma2=9801

>>>
```

2. Recapitulação

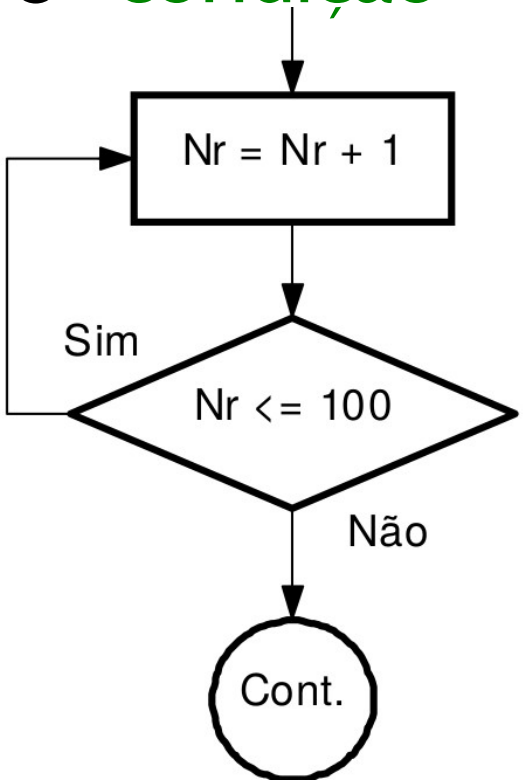
Códigos de formatação para o printf()	Significado
%c	Caractere simples
%d ou %i	Inteiro decimal com sinal
%e ou E	Notação científica (“e” minúsculo ou “E” maiúsculo)
%f	Ponto flutuante
%g	O mais curto entre: %e ou %f
%G	O mais curto entre: %E ou %f
%o	Inteiro octal sem sinal
%s	String de caracteres
%u	Inteiro sem sinal
%x ou %X	Inteiro hexadecimal sem sinal (minúsc. ou maiúsc.)
%%	Imprime o caractere “%”

2. Recapitulação

Repetições com testes no final:

faça <comandos> **enquanto** <condição>

```
Algoritmo ExemploFacaEnquantoContador
  VAR nr = 0
  FAÇA
      nr <- nr + 1
  ENQUANTO nr <= 100
Fim-Algoritmo
```



2. Recapitulação

Repetição com testes no final:

Exemplo 4: Escreva um programa que solicita ao usuário valores inteiros positivos e conta a quantidade de número pares e a quantidade de números ímpares digitados. O usuário deve continuar fornecendo novos valores até que algum número negativo seja digitado. O programa deve mostrar ao final quantos números pares e quantos números ímpares foram digitados.

2. Recapitulação

```
def main():
    contador_impares = 0
    contador_pares = 0
    while True:
        n = int(input("Escreva um novo valor:"))
        if n >= 0:
            if n % 2 == 0:
                contador_pares += 1
            else:
                contador_impares += 1
        else:
            break
    print(f"Voce digitou {contador_pares} numeros pares")
    print(f"Voce digitou {contador_impares} numeros impares")
```

2. Recapitulação

Shell

```
>>> %Run exercicio04.py
  Escreva um novo valor:0
  Escreva um novo valor:9
  Escreva um novo valor:3
  Escreva um novo valor:4
  Escreva um novo valor:2
  Escreva um novo valor:10
  Escreva um novo valor:2
  Escreva um novo valor:1
  Escreva um novo valor:-2
  Voce digitou 5 numeros pares
  Voce digitou 3 numeros impares

>>>
```

3. Iteração

Realizar tarefas repetitivas sem cometer erros é algo que os computadores fazem bem e as pessoas nem tanto. É por isso que os computadores são utilizados muitas vezes para automatizar esses tipos de tarefas.

3. Iteração

A execução repetida de uma sequência de instruções é chamada de **iteração** (*iteration*). Como iterar é muito comum, Python tem várias características para torná-la mais fácil. Nós já vimos o comando **`while`**, mas a forma mais comum é via comando **`for`**.

3. Iteração

```
print("digite o valor de N:")
n = int(input())
contador = 0
while contador < n:
    print(contador)
    contador += 1
```

```
print("digite o valor de N:")
n = int(input())
for contador in range(n):
    print(contador)
```

3. Iteração

O comando **`for`** processa cada item em uma lista. Cada item, por sua vez, é (re)atribuído a variável de iteração, e o corpo do laço é executado.

3. Iteração

```
for contador in range(start, end, step):
    pass
```

3. Iteração

Exemplo:

```
for seq in range(50,1000,100):
    print(seq)
```

Shell

```
>>> %Run -c $EDITOR_CONTENT
 50
 150
 250
 350
 450
 550
 650
 750
 850
 950

>>>
```

3. Iteração

Nesse caso, o comando **`range`** é um **iterador** (*iterator*).

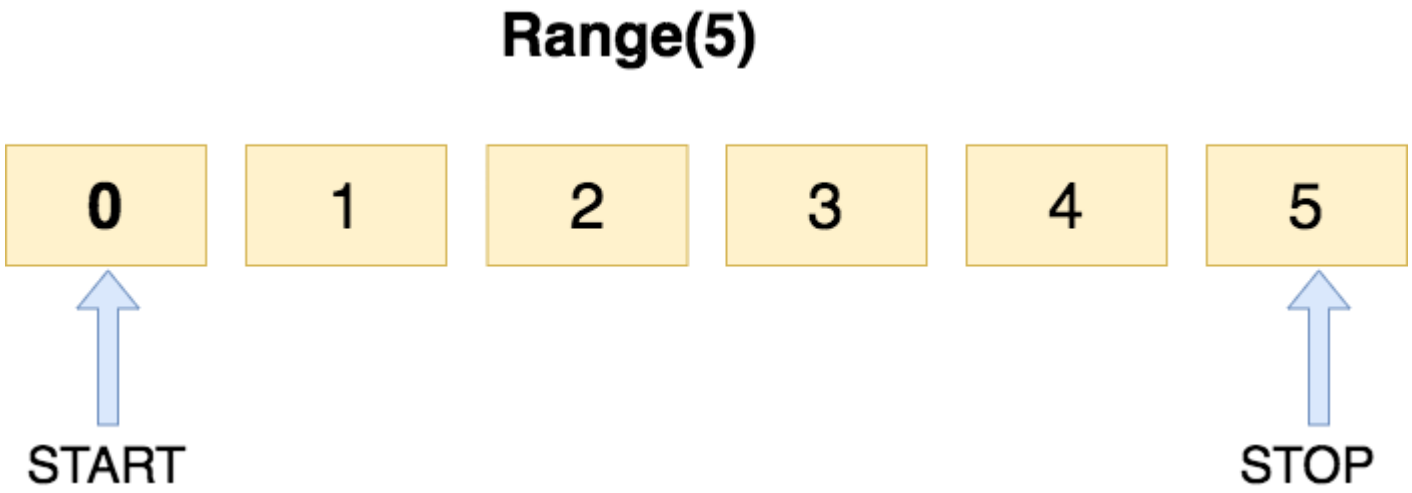
```
it = tuple(range(50,1000,100))
for seq in it:
    print(seq)
```

```
Shell
>>> %Run -c $EDITOR_CONTENT
 50
 150
 250
 350
 450
 550
 650
 750
 850
 950

>>>
```

3. Iteração

Nesse caso, o comando **`range`** é um **iterador** (*iterator*).



3. Iteração

Portanto, o comando **`for`** itera sobre os elementos de um objeto iterador.

```
for f in ["Amy", "Brad", "Carlos", "Daniel", "Ella"]:
    invitation = f"Hi, {f}. Please, come to my party!"
    print(invitation)
```

```
Shell
>>> %Run -c $EDITOR_CONTENT

  Hi, Amy. Please, come to my party!
  Hi, Brad. Please, come to my party!
  Hi, Carlos. Please, come to my party!
  Hi, Daniel. Please, come to my party!
  Hi, Ella. Please, come to my party!

>>>
```

3. Iteração

Curiosidade: para criar um iterador, você deve definir uma classe e implementar os métodos `__iter__()` e `__next__()` em seu objeto.

3. Iteração

- O método `__iter__()` retorna um iterador para o objeto fornecido (**list**, **sets**, **tuples**, etc). Ele cria um objeto que pode ser acessado um elemento por vez usando a função `__next__()`, que geralmente é útil ao lidar com *loops*.

3. Iteração

- O método `__next__()` também permite fazer operações, e deve retornar o próximo item da sequência.

3. Iteração

```
class MeusNumeros:
  def __iter__(self):
    self.a = 1
    return self

  def __next__(self):
    x = self.a
    self.a += 1
    return x

iteravel = MeusNumeros()
iterator = iter(iteravel)

print(next(iterator))
print(next(iterator))
print(next(iterator))
```

3. Iteração

- O exemplo anterior continuaria indefinidamente se você tivesse instruções **`next()`** suficientes ou se fosse usado em um *loop* **`for`**.
- Para evitar que a iteração continue para sempre, podemos usar a instrução **`StopIteration`**.

3. Iteração

```
Shell
>>> %Run -c $EDITOR_CONTENT
  1
  2
  3
  4
  5
  6
  7
  8
  9
  10
>>>
```

```
class MeusNumeros:
  def __iter__(self):
    self.a = 1
    return self

  def __next__(self):
    if self.a <= 10:
      x = self.a
      self.a += 1
      return x
    else:
      raise StopIteration

iteravel = MeusNumeros()
iterator = iter(iteravel)

for i in iterator:
    print(i)
```


THANK
YOU
GRACIAS
ARIGATO
SHUKURIA
DANKSCHEEN
TASHAKKURATU
YAQHANYELAY
SUKSAMA
EKHMET
BİYAN
SHUKRIA
TINGKI
GRAZIE
MEHRBANI
MAAKE
KOMAPSUMNIDA
GOZAIMASHITA
EFCHARISTO
JUSPAXAR
PALDIES
BOLZÏN
MERCI

Dúvidas?