MC-102 — Aula 05
Comandos Repetitivos

Eduardo C. Xavier

Instituto de Computação – Unicamp

30 de Setembro de 2020

Roteiro

Comandos Repetitivos

- Até agora vimos como escrever programas capazes de executar comandos de forma linear, e, se necessário, tomar decisões com relação a executar ou não um bloco de comandos.
- Entretanto, eventualmente é necessário executar um bloco de comandos várias vezes para se obter o resultado esperado.

Introdução

- Ex.: Programa que imprime todos os números de 1 até 4.
- Será que dá pra fazer com o que já sabemos?

```
print("1")
print("2")
print("3")
print("4")
```

Introdução

- Ex.: Programa que imprime todos os números de 1 até 100.

```
print("1")
print("2")
print("3")
print("4")
print("5")
    .
    .
    .
print("100")
```

Introdução

- Ex.: Programa que imprime todos os números de 1 até n (informado pelo usuário).

```
n = int(input('Digite um número:'))
print(1)
if(n>=2):
    print(2)
if(n>=3):
    print(3)
      .
      .
      .
if(n >= 100):
    print("100")
```

- Note que o programa é válido para $n \leq 100$.

Comando **while**

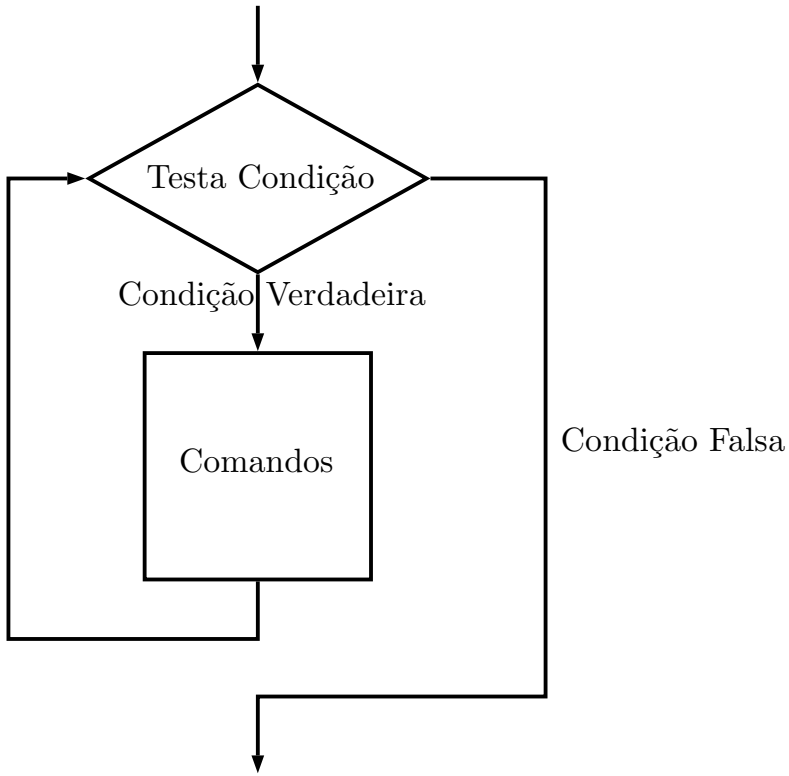
- Estrutura:

```
while condição :
    comandos
```

- Enquanto a condição for verdadeira (`True`), ele executa o(s) comando(s).

Comando **while**

- Passo 1: Testa a condição. Se a condição for verdadeira vai para o Passo 2.
- Passo 2.1: Executa os comandos.
- Passo 2.2: Volta para o Passo 1.



Comando **while**

Imprimindo os 100 primeiros números inteiros:

```
i = 1
while i <= 100:
    print(i)
    i = i + 1
```

Comando **while**

Imprimindo os *n* primeiros números inteiros:

```
n = int(input("Digite um número:"))
i = 1
while i <= n:
    print(i)
    i = i + 1
```

Comando **while**

- 1. O que acontece se a condição for falsa na primeira vez?

```
a = 1
while a!=a:
    a=a+1
```

- 2. O que acontece se a condição for sempre verdadeira?

```
a = 1
while a == a:
    a=a+1
```

Comando **while**

- 1. O que acontece se a condição for falsa na primeira vez?

```
a = 1
while a!=a:
    a=a+1
```

Resposta: Ele nunca entra no laço.

- 2. O que acontece se a condição for sempre verdadeira?

```
a = 1
while a == a:
    a=a+1
```

Resposta: Ele entra no laço e nunca sai (laço infinito).

Breve introdução à listas

- Uma lista em Python é uma estrutura que armazena vários dados que podem ser de um mesmo tipo ou não.
- O acesso a um dado específico da lista se da por indicação de sua posição.
- Uma lista é criada com a construção: $[\text{dado}_1, \text{dado}_2, \ldots, \text{dado}_n]$.

```
>>> a = [1, "ola", 2]
>>> type(a)
<class 'list'>
>>> a[0]
1
>>> a[1]
'ola'
>>> a[2]
2
```

O comando **for**

- Estrutura:

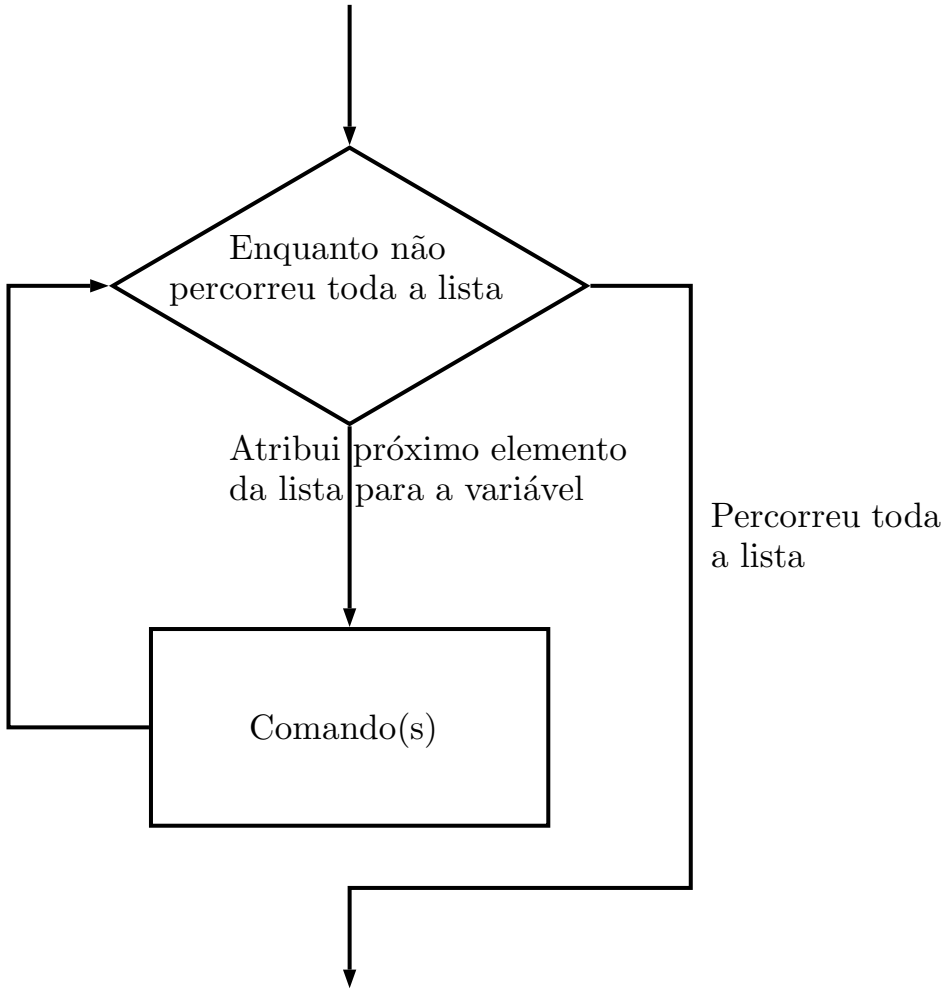
```
for variável in lista:
  comando(s)
```

- Para cada elemento da lista, em ordem de ocorrência, é atribuído este elemento à variável e então é executado o(s) comando(s).

O comando **for**

- Passo 1: Se não percorreu toda a lista, atribui-se próximo elemento da lista para a variável.
- Passo 2.1: Executa comandos.
- Passo 2.2: Volta ao Passo 1.

O comando **for**



O comando **for**

- O programa abaixo usa o laço **for** para imprimir números de uma lista.

```
a = [1, 2, 3]
for i in a:
  print(i)
```

A função **range**

- É comum fazermos um laço **for** iterar sobre valores numéricos.
- Em Python o comando **range(n)** gera uma lista[1] com os valores de 0 até $n - 1$.
- O programa abaixo imprime os números de 0 até 9.

```
for i in range(10):
    print(i)
```

[1] é um iterator na verdade, mas funciona como uma lista para nossos propósitos aqui

A função **range**

- Podemos especificar um intervalo de valores na função **range**:
  - **range(i, f)**: gera-se números de i até $f - 1$.
- O programa abaixo imprime os números de 5 até 9.

```
for i in range(5,10):
  print(i)
```

A função **range**

- Podemos especificar um passo a ser considerado no intervalo de valores da função **range**.
  - **range(i, f, p)**: gera-se valores a partir de i com incremento de p até $f - 1$.
- O programa abaixo imprime os números pares entre 0 e menores que 13.

```
for i in range(0,13,2):
  print(i)
```

O Comando **for**

Imprimindo os *n* primeiros números inteiros:

```
n = int(input("Digite um número:"))
for i in range(1,n+1):
    print(i)
```

Variável Acumuladora

- Vamos ver alguns exemplos de problemas que são resolvidos utilizando laços.
- Há alguns padrões de solução que são bem conhecidos, e são úteis em diversas situações.
- O primeiro padrão deles é o uso de uma “variável acumuladora".

Problema

Ler um inteiro positivo n, em seguida ler n números do teclado e apresentar a soma destes.

Soma de números

- Como n não é definido a priori, não podemos criar n variáveis e depois somá-las.
- A idéia e criar uma variável acumuladora que a cada iteração de um laço acumula a soma de todos os números lidos até então.
- Propriedade da **acumuladora**:
  - No início da i-ésima iteração tem a soma dos $(i-1)$ números lidos anteriormente.
  - No fim da i-ésima iteração terá a soma dos i números lidos (adiciona à seu valor o novo número lido).

Soma de números

- Como n não é definido a priori, não podemos criar n variáveis e depois somá-las.
- A idéia e criar uma variável acumuladora que a cada iteração de um laço acumula a soma de todos os números lidos até então.
- Propriedade da **acumuladora**:
  - No início da i-ésima iteração tem a soma dos $(i-1)$ números lidos anteriormente.
  - No fim da i-ésima iteração terá a soma dos i números lidos (adiciona à seu valor o novo número lido).
- Pseudo-código:

```
acumuladora = 0 # inicialmente não somamos nada
Repita n vezes
        Leia um número aux
        acumuladora = acumuladora + aux
```

Soma de números

- Para repetir a leitura de *n* números podemos usar tanto um laço **for**

```
n = int(input('Digite a quantidade de números:'))

for i in range(0,n):
    #executa comandos n vezes
```

ou um laço **while**

```
n = int(input('Digite a quantidade de números:'))

i = 1
while i <= n:
    #executa comandos n vezes
    i = i + 1
```

Soma de números

- Abaixo temos uma solução utilizando o comando **for**.

```
n = int(input('Digite a quantidade de números:'))
soma = 0
for i in range(0,n):
    aux = int(input('Número:'))
    soma = soma + aux

print('Soma é:', soma)
```

Calculando potências de 2

Mais um exemplo:

Problema

Leia um inteiro positivo n, e imprima as potências: $2^0, 2^1, \ldots, 2^n$.

Calculando potências de 2

- Usamos uma variável acumuladora que no início da i-ésima iteração de um laço, possui o valor 2^i.
- Imprimimos este valor e atualizamos a acumuladora para a próxima iteração, multiplicando esta variável por 2.
- Propriedade da **acumuladora**:
  - No início da i-ésima iteração tem o valor de 2^i que é impresso.
  - No fim da i-ésima iteração seu valor é atualizado para 2^{i+1} para a próxima iteração.

Calculando potências de 2

- Usamos uma variável acumuladora que no início da i-ésima iteração de um laço, possui o valor 2^i.
- Imprimimos este valor e atualizamos a acumuladora para a próxima iteração, multiplicando esta variável por 2.
- Propriedade da **acumuladora**:
  - No início da i-ésima iteração tem o valor de 2^i que é impresso.
  - No fim da i-ésima iteração seu valor é atualizado para 2^{i+1} para a próxima iteração.
- Pseudo-código:

```
acumuladora = 1 # Corresponde a 2^0
Para i=0 até n faça:
        imprima acumuladora
        acumuladora = acumuladora * 2
```

Calculando Potências de 2

Em Python:

```
n = int(input('Digite n:'))
i = 0
pot = 1 #corresponde a 2^0
while i <= n:
    print('2^%d = %d' %(i, pot))
    pot = pot * 2
    i = i + 1
```

OBS: Já vimos o uso de %f ou %.Nf para imprimir números float. Aqui usamos %d para números inteiros.

Calculando o valor de $n!$

Problema

Fazer um programa que lê um valor inteiro positivo n e calcula o valor de $n!$.

- Lembre-se que $n! = n * (n-1) * (n-2) * \ldots 2 * 1$.

Calculando o valor de $n!$

- Criamos uma variável acumuladora que no início da i-ésima iteração de um laço armazena o valor de $(i-1)!$.
- Durante a i-ésima iteração atualizamos a variável acumuladora multiplicando esta por i obtendo $i!$.
- Propriedade da **acumuladora**:
  - No início da i-ésima iteração tem o valor de $(i-1)!$.
  - No fim da i-ésima iteração seu valor é atualizado para $i! = (i-1)! * i$.
- No fim do laço, após n iterações, teremos na acumuladora o valor de $n!$.

Calculando o valor de $n!$

- Criamos uma variável acumuladora que no início da i-ésima iteração de um laço armazena o valor de $(i-1)!$.
- Durante a i-ésima iteração atualizamos a variável acumuladora multiplicando esta por i obtendo $i!$.
- Propriedade da **acumuladora**:
  - No início da i-ésima iteração tem o valor de $(i-1)!$.
  - No fim da i-ésima iteração seu valor é atualizado para $i! = (i-1)! * i$.
- No fim do laço, após n iterações, teremos na acumuladora o valor de $n!$.
- Pseudo-código:

```
acumuladora = 1 #corresponde a 0!
Para i=1 até n faça:
    acumuladora = acumuladora * i
    i = i + 1
```

Calculando o valor de $n!$

Em Python:

```
n = int(input('Digite n:'))
fat = 1 #corresponde a 0!
for i in range(1, n+1):
    #no fim do laço devemos ter i!
    fat = fat*i
print('Fatorial de',n,'é',fat)
```

Laços e o comando **break**

- O comando **break** faz com que a execução de um laço seja terminada, passando a execução para o próximo comando depois do final do laço.

```
for i in range (1,11):
    if(i >= 5):
        break
    print(i)
print("Terminou o laço")
```

O que será impresso?

Laços e o comando **break**

- O comando **break** faz com que a execução de um laço seja terminada, passando a execução para o próximo comando depois do final do laço.

```
for i in range (1,11):
    if(i >= 5):
        break
    print(i)
print("Terminou o laço")
```

O que será impresso?

Resposta: Os números de 1 até 4 e depois a frase “Terminou o laço".

Laços e o comando **break**

- Assim como a “condição"em laços, o comando **break** é utilizado em situações de parada de um laço.

Exe.: Imprimindo os números de 1 até 10.

```
i=1
while True:
    if(i > 10):
        break
    print(i)
    i = i+1
```

é equivalente a:

```
i=1
while i<=10:
    print(i)
    i = i+1
```

Laços e o comando **continue**

- O comando **continue** faz com que a execução de um laço seja alterada para final do laço.

```
for i in range(1,11):
    if i==5:
        continue
    print(i)

print('Terminou o laço')
```

O que será impresso?

Laços e o comando **continue**

- O **continue** faz com que a execução de um laço seja alterada para final do laço.

```
for i in range(1,11):
    if i==5:
        continue
    print(i)

print('Terminou o laço')
```

O que será impresso?
Resposta: Os números de 1 até 10, exceto o número 5, e depois a frase "Terminou o laço".

Laços e o comando **continue**

- O **continue** é utilizado em situações onde comandos dentro do laço só devem ser executados caso alguma condição seja satisfeita.

Exe.: Imprimindo área de um círculo, mas apenas se raio for par (e entre 1 e 10).

```
for r in range(1,11):
    if( r % 2  != 0): #se o número for ímpar pulamos
        continue
    area = 3.1415*r*r
    print("%.2f" %area)
```

Mas note que poderíamos escrever algo mais simples:

```
for r in range(2,11,2):
    area = 3.1415*r*r
    print("%.2f" %area)
```

Exercício

- Faça um programa que imprima um menu de 4 pratos na tela e uma quinta opção para sair do programa. O programa deve imprimir o prato solicitado. O programa deve terminar quando for escolhido a quinta opção.

Exercício

- Faça um programa que lê dois números inteiros positivos a e b. Utilizando laços, o seu programa deve calcular e imprimir o valor a^b.

Exercício

- Faça um programa que lê um número n e que computa e imprima o valor

$$\sum_{i=1}^{n} i.$$

OBS: Não use fórmulas como a da soma de uma P.A.

Exercício

- Faça um programa que lê um número n e imprima os valores entre 2 e n que são divisores de n.

Exercício

- Faça um programa que lê um número n e imprima os valores

$$\sum_{i=1}^{j} i$$

para j de 1 até n, um valor por linha.