

Prática de Programação em Sistemas de Informação

Comandos de Decisão e Repetição





Desenvolvimento do material

Alexandre Louzada

1ª Edição

Copyright © 2022, Unigranrio

Nenhuma parte deste material poderá ser reproduzida, transmitida e gravada, por qualquer meio eletrônico, mecânico, por fotocópia e outros, sem a prévia autorização, por escrito, da Unigranrio.

Sumário

Comandos de Decisão e Repetição

Para início de conversa...	3
Objetivos	3
1. Uso de Comandos de Decisão	4
2. Uso de Comandos de Repetição	8
3. Uso do Break e do Continue	11
Referências	14



Para início de conversa...

Neste capítulo, aprenderemos a utilizar, em programas de computadores, as estruturas de controle condicionais e de repetição.

As estruturas condicionais (ou de seleção) são utilizadas para selecionar, mediante comparação de dados, quais instruções de um programa devem ou não ser executadas. Já as estruturas de repetição (ou laços) são utilizadas para que uma ou mais instruções sejam executadas repetidas vezes em um programa.

Objetivos

- Criar programas que possuam a capacidade de selecionar quais blocos de códigos devem ou não ser executados.
- Criar programas que manipulem iterações em que um bloco de código seja executado repetidas vezes.



1. Uso de Comandos de Decisão

As estruturas de controle condicionais são aquelas que selecionam quais instruções de um programa vão ser executadas e quais não serão executadas. E isso é realizado por meio de testes condicionais, geralmente criados com os operadores relacionais.

A linguagem Python utiliza o comando **if** para realizar os testes condicionais. Para isso, o programador pode utilizar seleção simples, seleção composta ou seleção aninhada de acordo com sua necessidade. Veja que o comando abaixo de **if** possui uma tabulação (geralmente quatro espaços). Esse espaçamento, também chamado de indentação, é **fundamental** quando utilizamos estruturas de controle em Python:

```
if condição:  
    instrução que será executada se a condição for verdadeira
```

No exemplo a seguir, temos um trecho de código que demonstra como trabalhar com seleção simples em Python testando o valor de uma idade e, dependendo do resultado lógico, exibir uma mensagem.

```
if idade >= 18:  
    print("Maior de Idade")
```

Vamos observar a estrutura de seleção simples (**if**) sem bloco de código no contexto de um programa em Python.

```
print("Verifica a idade ")  
idade = int(input("Digite a sua idade:"))  
  
if idade>= 18:  
    print("Maior de idade")
```

Vamos observar, agora, o mesmo exemplo anterior, mas, agora, com bloco de código. Quando a condição for verdadeira, além de imprimir a mensagem “Maior em idade”, também irá calcular e imprimir a quantos anos o indivíduo já é maior em idade, caso contrário, nada será executado.

```
print("Verifica a idade ")  
idade = int(input("Digite a sua idade:"))  
  
if idade>= 18:  
    print("Maior de idade")  
    print("Já é maior faz ", idade-18," Anos")
```

A estrutura de controle para seleção **if-else** possui uma condição e um bloco de instrução. Se a condição for verdadeira, um bloco de instrução é executado; se a condição for falsa, o outro bloco de instrução é executado. Vamos observar a sintaxe da estrutura de seleção composta (**if-else**) sem bloco de código:

```
if condição:  
    instrução que será executada se a condição for verdadeira  
else:  
    instrução que será executada se a condição for falsa
```

Observe, na sintaxe, que o código marcado em **vermelho** e que está logo abaixo do comando **if** será executado caso a condição seja verdadeira, descartando, assim, a execução do código marcado em **azul**. Por outro lado, o código marcado em **azul**, que está logo abaixo do comando **else**, será executado apenas se a condição for falsa, descartando, assim, a execução do código marcado em **vermelho**.

Vamos observar um exemplo de uma estrutura de seleção composta (**if-else**) que possui uma condição que avalia se o conteúdo de uma variável chamada `idade` é maior ou igual a 18. Caso a condição seja verdadeira, ou seja, o conteúdo da variável `idade` seja maior ou igual a 18, imprima a mensagem “Maior em Idade”, caso contrário, ou seja, caso a condição seja falsa, imprima a mensagem “Menor em idade”.

```
if idade >= 18:  
    print("Maior em Idade");  
  
else:  
    print("Menor em Idade");
```

Observe que, no exemplo anterior, a instrução marcada em **vermelho**, que é a impressão da mensagem “Maior em Idade” e que está logo abaixo do comando **if**, será executada caso a condição seja verdadeira, ou seja, se o valor da variável `idade` for maior ou igual a 18, descartando, assim, a execução da instrução marcada em **azul**. Por outro lado, a instrução marcada em **azul**, que é a impressão da mensagem “Menor em Idade” e que está logo abaixo do comando **else**, será executada apenas se a condição for falsa, ou seja, se o valor da variável `idade` não for maior ou igual a 18, descartando, assim, a execução da instrução marcada em **vermelho**.

Vamos observar a estrutura de seleção composta (**if-else**) sem bloco de código no contexto de um programa em Python.

```
print("Verifica a idade ")  
idade = int(input("Digite a sua idade:"))  
  
if idade>= 18:  
    print("Maior em idade")  
  
else:  
    print("Menor em idade")
```

Vamos observar, agora, o mesmo exemplo anterior, mas com bloco de código. Quando a condição for verdadeira, além de imprimir a

mensagem “Maior em idade”, também irá calcular e imprimir a quantos anos o indivíduo já é maior em idade; caso contrário, além de imprimir a mensagem “Menor em idade”, vai calcular e imprimir quantos anos faltam para o indivíduo se tornar maior em idade.

```
if idade >= 18:  
    print("Maior em idade")  
    print("Já é maior em idade a ", idade-18," anos")  
  
else:  
  
    print("Menor em idade")  
    print("Faltam ", 18-idade," anos para ser maior em idade")
```

Observe no exemplo anterior que, no código marcado em **vermelho**, temos o bloco de instruções que será executado caso a condição seja verdadeira. Observe que, como temos mais de uma instrução, eles ficam agrupados em um bloco de código associado ao comando **if**. Observe, também, que o código marcado em **azul** representa o bloco de instruções que será executado se a condição for falsa. Como também possui mais de uma instrução, elas também estão agrupadas em um bloco de código, mas este associado ao comando **else**.

Vamos observar a estrutura de seleção composta (**if-else**) com bloco de código no contexto de um programa em Python.

```
print("Verifica a idade ")  
idade = int(input("Digite a sua idade:"))  
  
if idade>= 18:  
    print("Maior em idade")  
    print("Já é maior em idade a ", idade-18," anos")  
  
else:  
    print("Menor em idade")  
    print("Faltam ", 18-idade," anos para ser maior em idade")
```

As estruturas de seleção aninhadas é o recurso de se colocar uma estrutura de seleção dentro de outra de forma encadeada, formando um ninho de estruturas. Esse recurso é utilizado quando existem mais de duas saídas a serem avaliadas. Vamos observar a sintaxe das estruturas de seleção aninhadas:



```

# Sintaxe sem bloco de código

if condição:
    instrução que será executada se a condição for verdadeira
elif condição:
    instrução que será executada se a condição anterior for falsa e a condição
    atual for verdadeira
elif condição:
    instrução que será executada se a condição anterior for falsa e a condição
    atual for verdadeira
... // é possível colocar diversos ninhos (else if)
Else:
    instrução que será executada se todas as condições forem falsas

# Sintaxe com bloco de código

if condição
    instruções que serão executadas se a condição for verdadeira
elif condição:
    instruções que serão executadas se a condição anterior for falsa e a
    condição atual for verdadeira
elif condição:
    instruções que serão executadas se a condição anterior for falsa e a
    condição atual for verdadeira
... // é possível colocar diversos ninhos (elif)
else {
    instruções que serão executadas se todas as condições forem falsas

```

Vamos observar um exemplo de estruturas de seleção aninhadas que possuem condições que avaliam se o conteúdo de uma variável chamada de **numero** é maior, menor ou igual a 0 (zero). Caso o conteúdo da variável **numero** seja maior que 0 (zero), imprima a mensagem “Número Positivo”; caso o conteúdo da variável **numero** seja menor que 0 (zero), imprima a mensagem “Número negativo” e, caso não seja maior e nem menor que 0 (zero), imprima a mensagem “Número igual a zero”.

```

if numero > 0:
    print("Número Positivo")
elif numero < 0:
    print("Número Negativo")
else:
    print("Número igual a zero")

```

Observe, no exemplo anterior, que a instrução marcada em **vermelho** será executada se a condição do primeiro **if** for verdadeira, descartando, assim, a execução das instruções marcadas em **azul** e **verde**. A instrução marcada em **azul** será executada se a condição do primeiro **if** for falsa e a condição do segundo **if** for verdadeira, descartando, assim, a execução das instruções marcadas em **vermelho** e **verde**. A instrução marcada em **verde** será executada quando a condição de todos os **ifs** for falsa, descartando, assim, a execução das instruções marcadas em **vermelho** e **azul**.

Vamos observar as estruturas de seleção aninhadas no contexto de um programa em Python.

```
numero = int(input("Digite um numero:"))

if numero > 0:
    print("Numero Positivo")
elif numero < 0:
    print("Numero Negativo")
else:
    print("Numero igual a zero")
```

2. Uso de Comandos de Repetição

Em Python, o comando **for** pode ser utilizado para repetir uma ou mais instruções por uma quantidade determinada de vezes. Existem diferentes modos de utilizar esse comando e vamos explorar a maior parte deles na próxima unidade. Por hora, vamos ver um dos modos de utilizar o comando **for**:

Vamos observar um exemplo de uma estrutura de repetição **for** que imprime 10 vezes a mensagem “Seja bem-vindo(a)”.

```
for x in range(10):
    print("Seja bem-vindo(a)")
```

Vamos observar a estrutura de repetição **for** sem bloco de código no contexto de um programa em Python, em um programa que realiza a leitura de um número inteiro e imprime todos os números entre 1 e o número lido.

```
n = int(input("Digite um numero:"))

n+=1

for x in range(1,n):
    print("Numero :", x)
```

Vamos observar um exemplo de uma estrutura de repetição **for** que realiza a leitura de 5 números e imprime o triplo de cada número lido:

```
for x in range(5):
    numero = int(input("Digite um numero:"))

    print("Triplo :", numero*3)
```

Vamos observar a estrutura de repetição **for** com bloco de código no contexto de um programa em Python, em um programa que leia 10 números inteiros, calcule e imprima a soma de todos eles:

```
soma = 0

for x in range(10):

    numero = int(input("Digite um número:"))

    soma+=numero

print("Soma dos números :", soma)
```

Observe o exemplo acima, em que, no código marcado em **vermelho**, temos a indentação do comando **print**, indicando que ele não está subordinado à estrutura de repetição **for**.

Em Python, o comando **while** pode ser utilizado para repetir uma ou mais instruções por uma quantidade indeterminada de vezes. Nessa estrutura, o resultado do teste condicional determina o número de vezes que a repetição ocorre.

Vamos ver, agora, a forma mais simplificada de uso do **while** em Python:

```
while condição:

    instrução
```

Podemos utilizar o comando **while** com ou sem um bloco de código (um agrupamento de instruções em uma estrutura de controle).

Vamos observar, agora, um exemplo prático. No código abaixo, uma mensagem será exibida uma determinada quantidade de vezes:

```
cont = 1

while cont <= 10:
    print("Programação Python")
    cont+=1
```

Analisando o exemplo, vemos que uma variável **cont** é inicializada com o valor 1. Na próxima linha de código, o comando **while** é utilizado testando o valor da variável **cont**. Enquanto a variável **cont** for menor ou igual a 10, os comandos subordinados ao **while** serão executados, ou seja, será exibida a mensagem “Programação Python” e a variável **cont** receberá seu próprio valor mais um.

Vamos observar um outro exemplo em Python, em que um programa realiza a leitura de um número inteiro e imprime todos os números entre 1 e o número lido:

```
numero = int(input("Digite um número:"))

cont = 1

while cont <= numero:
    print(cont)
    cont+=1
```

No exemplo acima, o número de repetições é controlado pelo comando **while** com o auxílio das variáveis **cont** e **numero**. Enquanto o conteúdo da variável **cont** for menor ou igual ao conteúdo da variável **numero**, os comandos subordinados ao while serão executados, ou seja, será exibido o valor da variável **cont** e, em seguida, terá seu valor incrementado em 1.

Vamos observar um exemplo de um programa que realiza a leitura de uma quantidade indeterminada de números e exibe o triplo de cada número lido.

```
numero=1

while numero != 0:
    numero = int(input("Digite um numero:"))
    print("Triplo :", numero*3)
```

Vamos observar a estrutura de repetição **while** com bloco de código no contexto de um programa em Python, em um programa que faz a leitura de uma quantidade indeterminada de números inteiros positivos, calcula e exibe a soma de todos eles. *Flag* de saída: número lido negativo:

```
numero = int(input("Digite um numero:"))

soma=0

while numero > 0:
    soma+=numero
    numero = int(input("Digite um numero:"))

print("Soma dos números",soma)
```

No exemplo acima, a entrada de dados é realizada antes da estrutura de repetição para que, na primeira vez em que a condição for avaliada, a variável **numero** já possua um valor para ser comparado, caso seja maior que zero. O comando **while** é que irá executar um número indeterminado de vezes às instruções, dentro do seu bloco de código. A variável **soma** é utilizada para acumular cada número lido, em que a leitura do primeiro número é realizada fora da estrutura de repetição. Na linha seguinte, temos a leitura de um número para voltar à condição e avaliar, novamente, se as instruções do bloco de código vão ou não ser executadas. Mais uma vez, lembramos a importância da **indentação** para que o programa funcione da maneira esperada.

3. Uso do Break e do Continue

Os comandos **break** e **continue** são recursos das estruturas de repetição que permitem a interrupção de um único ciclo ou do laço de repetição. É importante saber a diferença entre elas.

O objetivo do comando **break** é fornecer a capacidade de forçar a interrupção da execução de uma repetição. Por exemplo, com a instrução **break**, podemos interromper a repetição, ainda que se uma condição **while** seja verdadeira:

```
"""
Exemplo
Saia do loop quando eu tiver 3:
"""

i = 1

while i < 6:
    print(i)
    if i == 3:
        break
    i += 1

# Vai exibir os números 1, 2 e 3
```

O comando **continue** interrompe a execução do ciclo sem interromper a execução do laço de repetição. No exemplo a seguir, a instrução **continue** vai parar uma iteração específica e continuar com a próxima:

```
"""
Exemplo
Continue para a próxima iteração se eu for 3::

"""

i = 0

while i < 6:
    i += 1
    if i == 3:
        continue
    print(i)

# Vai exibir os valores 1,2,4,5 e 6
```

Agora que aprendemos os principais aspectos relacionados às estruturas de condição e repetição, vamos testar alguns programas que mostram como elas podem ser muito úteis para resolução de problemas. Vamos aproveitar para utilizar alguns recursos que aprendemos na Unidade

de Aprendizagem 1 como uso de operadores e formatação de valores exibidos na tela:

```
"""
Esse programa exibe a tabuada de multiplicação de um numero inteiro
de 1 a 10

"""

numero = int(input("digite um numero inteiro de 1 a 10: "))

cont = 1

if numero >=1 and numero <=10:

    while cont <= 10:

        resultado = numero * cont

        print ("%d X %d = %d" % (numero, cont, resultado))

        cont = cont + 1

else:

    print ("apenas valores de 1 a 10")
```

```
"""
Esse programa exibe a conversão de dólar para real nos
valores de 1 a 99 dólares"

"""

real = float(input("digite o valor de um dólar em real: "))

for dolar in range(1,100):

    print ("US$ %.2f = R$ %.2f" % (dolar,dolar*real))

    print ("-" * 25)
```

```
"""
```

Esse programa permite a digitação de valores inteiros maiores do que zero e, ao final, informa a soma dos valores digitados, a média, o maior e o menor número.

```
"""
```

```
conta = 1
soma=maior_numero=0

print("Digite números inteiros positivos")

numero = int(input("Digite o %dº número: << % conta))

menor_numero=numero

while numero > 0:

    soma += numero
    if numero > maior_numero:
        maior_numero = numero
    if numero < menor_numero:
        menor_numero = numero
    conta+=1
    numero=int(input("Digite o %dº número: << % conta))

media = soma / (conta-1)
print("Soma:",soma)
print("Maior:", maior_numero)
print("Menor:", menor_numero)
print("Media: %.2f" % media)
print("O %dº número não foi considerado" % conta)
```

```
"""
```

Elabore um programa que leia o código e o preço de compra de 5 produtos. O programa deve avaliar a classificação do preço de venda do produto, baseado no seu preço de compra, de seu valor de imposto (35%) e de sua margem de lucro (40%). O programa deve apresentar ao usuário a informação sobre se o preço de venda do produto estava barato, normal ou caro. Os critérios para classificar o preço de venda são:

```
preço de venda menor que 5% do valor do salário mínimo - Barato
preço de venda entre 5% e 15% do valor do salário mínimo - Normal
preço de venda maior que 15% do valor do salário mínimo - Caro
"""
```

```
# Valor-base para o salário mínimo
sal_minimo = 1000.00
```

```
for i in range (5):
    print("ENTRADA DE DADOS DO PRODUTO")
```

```
codigo = int(input("Digite o código do produto: "))
precomp=float(input("Digite o preço de compra: R$ "))
```

```
imp = precomp * 35 / 100          # cálculo do imposto
lucro = precomp * 40 / 100         # cálculo do percentual de lucro
prevend = precomp + imp + lucro   # cálculo do preço de venda
```

```
print("LISTA DOS DADOS DO PRODUTO")
```

```
print("+" ,"-*70, "+" )
      print("| Código | Preço Compra | Imposto | Lucro | Preço Venda |"
Classificacao |")
      print("+" ,"-*70, "+" )
```

```
if prevend < sal_minimo * 0.05:
    print("| %6d | %12.2f | %7.2f | %5.2f | %11.2f | %-13s |" % (codigo,
precomp, imp, lucro, prevend, "Barato"))
elif prevend >= sal_minimo * 0.05 and prevend < sal_minimo * 0.15:
    print("| %6d | %12.2f | %7.2f | %5.2f | %11.2f | %-13s |" % (codigo,
precomp, imp, lucro, prevend, "Normal"))
else:
    print("| %6d | %12.2f | %7.2f | %5.2f | %11.2f | %-13s |" % (codigo,
precomp, imp, lucro, prevend, "Caro"))

print("+" ,"-*70, "+" )
```

De fato, é muito importante praticar para assimilar os conceitos obtidos até aqui. Ao digitar e executar esses exemplos, percebemos mais facilmente como utilizar as estruturas de condição e repetição em **Python**.

Nesta unidade, aprendemos sobre o uso das estruturas de condição para realizar comparações de dados e executar instruções de acordo com o resultado lógico dessas comparações. Essas estruturas (também chamadas de estruturas condicionais ou de decisão) podem ser aplicadas em pseudocódigo por meio do comando **if**.

Vimos, também, sobre o uso de estruturas de repetição (também chamada de iteração ou laço), que são uma estrutura presentes em linguagens de programação para repetir uma ou mais instruções necessárias para cumprir uma tarefa. Com seu uso, um programa pode ser construído com maior facilidade, além de torná-lo mais eficiente.

Um programador pode desenvolver um programa em Python para que uma repetição ocorra por uma quantidade indeterminada de vezes, até que uma condição seja satisfeita utilizando o comando **while** ou, ainda, construir uma repetição usando uma variável para controlar o número de repetições a ser realizada por meio do comando **for**.

Para utilizar essas estruturas de controle, é muito importante usar a indentação (espaçamento ou tabulação) para determinar quais instruções farão parte da estrutura que utilizarmos.

Vimos, também, como podemos utilizar os comandos **break** e **continue** para alterar a forma como as instruções serão executadas dentro das estruturas de repetição.

Referências

PERKOVIC, L. **Introdução à computação usando Python**: um foco no desenvolvimento de aplicações. Rio de Janeiro: LTC, 2016.