Introdução à programação em Python: comandos de repetição

Neste notebook você vai aprender como fazer para executar um trecho do seu programa várias vezes.

Isso é muito útil quando o problema que estamos tentando resolver envolve tarefas repetitivas!

Tarefas repetidas

Vamos começar com um exemplo simples: suponha que você queira imprimir os números de **1** a **5**.

Já vimos como fazer isso com uma lista:

In [1]:
```
print(list(range(1,6)))
```

```
[1, 2, 3, 4, 5]
```

Mas, e se quisermos imprimir um número por linha?

Nesse caso, você poderia fazer o seguinte:

In [2]:
```
print(1)
print(2)
print(3)
print(4)
print(5)
```

```
1
2
3
4
5
```

Agora, e se você quisesse imprimir os números de **1** a **100**, um por linha?
Tudo bem, imprimir os números de **1** a **100** com um monte de linhas `print()` é muito tedioso, mas também é factível.

O problema complica mesmo se você quisesse imprimir os números de **1** a n, onde n é um número que você obteve do usuário através de uma chamada do `input()`.

Isso não seria nem chato nem tedioso, seria impossível!

Para lidar com esse tipo de situação que exige repetição de tarefas, as linguagens de programação possuem estruturas de repetição (ou laços), como os comandos **`for`** e **`while`**.

Veja a seguir como você pode fazer para imprimir os números de 1 a 5 com o comando **`for`**:

In [3]:
```
for n in [1, 2, 3, 4, 5]:
    print(n)
```

```
1
2
3
4
5
```

In [4]:
```
for n in range(1,6):
    print(n)
```

```
1
2
3
4
5
```

O que ambos os códigos acima estão fazendo?

Eles estão iterando sobre os valores da lista: para cada valor distinto que está na lista, a variável `n` assume um desses valores. O corpo do laço está apenas mandando imprimir o valor que está nessa variável.

Laços do tipo `for` em Python servem precisamente para isso: iterar em elementos de uma lista.

Como vimos na primeira aula, listas podem armazenar qualquer tipo de valor. Assim, podemos ter o seguinte tipo de construção:

In [5]:
```
for e in ['gato', 'cachorro', True, 34]:
    print(e)
```

```
gato
cachorro
True
34
```

Podemos também interagir com o usuário:

In [1]:
```
#limite = int(input("Digite um número maior do que 1: "))
#for n in range(1, limite + 1):
#    print(n)

limite = int(input("Digite um número maior do que 5: "))
for n in range(5, limite + 1, 3):
    print(n)
```

```
Digite um número maior do que 5: 20
5
8
11
14
17
20
```

Faça você mesmo!

Modifique o código acima para que ele também peça o número inicial para o usuário. Execute uma vez de modo que ele imprima os números de **10** a **50**.

O comando `for`

Como você já deve ter percebido, o comando `for` faz com que uma variável receba, repetidamente, valores de uma dada lista e, para cada um desses valores, executa o bloco de código indentado logo abaixo dele.

A estrutura do comando `for` é a seguinte:

```
for variável in lista:
    comando
    comando
    etc.
restante do programa
```

Assim como no comando `if` , a primeira linha de código não indentada depois do comando marca o fim do bloco de código associado.

Exemplo

Suponha que queremos calcular a soma dos números de 1 a n.

Por exemplo, se $n = 5$, queremos calcular $1 + 2 + 3 + 4 + 5 = 15$.

Na verdade, existe uma fórmula fechada para essa soma, mas suponha que queremos fazer isso usando o comando `for` :

In [11]:
```
n = int(input("Digite o valor de n: "))
total = 0
for i in range(1, n + 1):
    total = total + i
print(total)
```

```
Digite o valor de n: 8
36
```

Atenção: Por que a segunda linha do código acima é `total = 0` ?

Faça você mesmo!

Modifique o código abaixo para calcular a soma dos quadrados dos números de 1 até n. Ou seja, quando o usuário digitar um número n, o programa deverá mostrar a soma

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

In [3]:
```
n = int(input("Digite o valor de n: "))
total = 0
for i in range(1, n + 1):
    total = total + i**2
print(total)
```

```
Digite o valor de n: 4
30
```

Exemplo

Também podemos utilizar um laço do tipo `for` em Python quando queremos realizar um bloco de comandos um certo número n de vezes.

Para isso, podemos iterar sobre uma lista que tenha tamanho n.

In [12]:
```
n = int(input("Digite a quantidade de repetições desejadas: "))
for i in range(n):
    print("Hello world")
```

```
Digite a quantidade de repetições desejadas: 5
Hello world
Hello world
Hello world
Hello world
Hello world
```

Note que não importa qual o valor de `i` em cada iteração (apesar de que sabemos que ele assumirá valores `0`, `1`, ..., `n-1`), apenas importa que ele assume `n` valores diferentes, que é a quantidade de vezes que queremos repetir os comandos.

Exemplo

O código a seguir pede que o usuário digite o número de vendas e um valor para cada venda. Depois, o programa calcula e imprime o valor total das vendas.

In [9]:
```
n = int(input("Qual o número de vendas? "))
total = 0.0
for i in range(n):
    valor = float(input("Qual o valor da venda? "))
    total = total + valor
    media = total / n
print("A total das vendas é:", total)
print("Média de todas as vendas: ",media)
```

```
Qual o número de vendas? 3
Qual o valor da venda? 70
Qual o valor da venda? 70
Qual o valor da venda? 70
A total das vendas é: 210.0
Média de todas as vendas:  70.0
```

Faça você mesmo!

Modifique o código acima de modo que, além do valor total das vendas, o programa imprima também o valor médio das vendas.

Importante

Assim como nos comandos `if`, um bloco de comandos ligados a um comando `for` pode conter quaisquer um dos comandos que vimos até agora, incluindo outros comandos `for`.

No exemplo a seguir, queremos contar quantos números digitados pelo usuário são pares.

In [4]:
```
n = int(input("Digite a quantidade total de números: "))
qtd_pares = 0
for i in range(n):
    novo_numero = int(input("Digite um número da sequência: "))
    if novo_numero % 2 == 0:
        qtd_pares = qtd_pares + 1
print("A quantidade total de números pares foi", qtd_pares)
```

```
Digite a quantidade total de números: 5
Digite um número da sequência: 2
Digite um número da sequência: 3
Digite um número da sequência: 4
Digite um número da sequência: 5
Digite um número da sequência: 6
A quantidade total de números pares foi 3
```

Faça você mesmo!

Modifique o código abaixo de modo que, toda vez que o usuário digitar uma venda de valor maior do que 100, o programa mostre **naquele mesmo momento** uma mensagem de parabéns para o vendedor.

Faça com que seu programa conte quantas vendas ultrapassaram R$ 100 e, no final, imprima uma mensagem informando essa quantidade.

In [10]:
```
n = int(input("Qual o número de vendas? "))
total = 0.0
vendMaior100 = 0.0
for i in range(n):
    valor = float(input("Qual o valor da venda? "))
    total = total + valor
    media = total / n
    if (valor > 100):
        print("Parabéns")
        vendMaior100 = vendMaior100 + 1
print("O total das vendas é:", total)
print("Média das vendas realizadas: ",media)
print("Você realizou",vendMaior100,"vendas acima de R$ 100,00.")
```

```
Qual o número de vendas? 6
Qual o valor da venda? 50
Qual o valor da venda? 60
Qual o valor da venda? 110
Parabéns
Qual o valor da venda? 120
Parabéns
Qual o valor da venda? 70
Qual o valor da venda? 200
Parabéns
O total das vendas é: 610.0
Média das vendas realizadas:  101.66666666666667
Você realizou 3.0  vendas acima de R$ 100,00.
```

O comando while

Agora vamos ver um outro comando para fazer repetições.

O nome desse comando é `while`, que quer dizer *enquanto* em português.

Vejamos um exemplo.

In [ ]:
```
n = 1
while n <= 5:
    print(n)
    n = n + 1
```

O que exatamente o nosso código está fazendo?

O primeiro passo é colocar o valor 1 em uma variável de nome `n`. O segundo passo é verificar a condição `n <= 5`. Se essa condição for verdadeira (e, no caso, é), então o conteúdo indentado logo abaixo do comando é executado.

No conjunto de comandos, temos a impressão do valor de `n` seguida do aumento no valor de `n` em uma unidade. O próximo passo do programa é verificar novamente a condição `n <= 5`.

Novamente, se a condição for verdadeira, o conteúdo indentado logo abaixo do comando while é executado.

Essa repetição se mantém até o primeiro momento em que a condição fique falsa. Nesse caso, quando `n` armazena o valor 6 a condição `n <= 5` fica falsa.

Quando a condição do while fica falsa, o conteúdo indentado logo abaixo dele não é mais executado e o programa segue para o próximo comando não indentado.

O comando **`while`** tem sempre a seguinte estrutura:

```
while condição:
    código indentado
    código indentado
    etc.
restante do programa
```

Observe que o comando **`while`** tem uma condição e um bloco de código associado que deve vir *indentado*.

No nosso exemplo acima, a condição é `n <= 5` e o bloco de código é

```
print(n)
n = n + 1
```

O comando **`while`** executa o bloco de código indentado enquanto a condição dada for verdadeira.

Faça você mesmo!

Modifique o código acima para que ele imprima os números de **10 a 50**.

Atenção: sempre tome muito cuidado quando você for usar o comando **`while`** pois, se acontecer da condição ser sempre verdadeira, seu programa irá ***entrar em loop infinito***, ou seja, ele ficará rodando "para sempre" e irá travar o kernel do Python!

Faça você mesmo!

O código a seguir imprime os números de 1 a 10. Modifique **apenas** o bloco de código do **`while`** de modo que ele imprima o seguinte:

```
 1 é ímpar
2 é par
3 é ímpar
4 é par
5 é ímpar
etc. (até 10)
```

Dica: use o comando **`if`** dentro do bloco de código do **`while`** .

In [ ]:
```
n = 1
while n <= 10:
    print(n)
    n = n + 1
```

O **`while`** é usado no lugar do **`for`** quando não sabemos quantas vezes devemos repetir um bloco de código.

Suponha novamente que queremos saber quantos números pares o usuário digitou, mas que dessa vez ele não sabe quantos números vai digitar.

Dessa vez, vamos pedir para o usuário digitar a palavra "fim" quando ele quiser parar de digitar números.

Veja a solução a seguir.

In [ ]:
```
qtd_pares = 0
# É possível (talvez improvável) que o usuário não queira digitar número nenhum
# Assim, ele pode digitar a palavra fim logo no início
dado = input("Digite um número da sequência ou a palavra 'fim': ")
# Se o que ele digitou não for a palavra fim, então executaremos o laço para pedir novos números
# Esse laço vai executar enquanto o usuário não digitar a palavra "fim"
while dado != "fim":
    # Se o programa está aqui dentro do laço, é porque a variável "dado" não tem a palavra fim
    # Ou seja, ela tem um número
    novo_numero = int(dado)
    # e agora podemos testar se esse número é par ou não
    if novo_numero % 2 == 0:
        qtd_pares = qtd_pares + 1
    # Agora precisamos pedir por uma nova informação do usuário
    dado = input("Digite um número da sequência ou a palavra 'fim': ")

print("A quantidade total de números pares foi", qtd_pares)
```

O código abaixo imprime "SENHA CORRETA" se o usuário digita 1234 e "SENHA INCORRETA" se o usuário digita qualquer outro número.

Note como ele fica executando até o usuário acertar a senha!

In [ ]:
```
senha_estah_errada = True
while senha_estah_errada:
    senha = int(input("Digite a sua senha numérica: "))
    if senha == 1234:
        print("SENHA CORRETA")
        senha_estah_errada = False
    else:
        print("SENHA INCORRETA")
```

Note que o mesmo comportamento pode ser replicado colocando-se a linha `print("SENHA CORRETA")` para fora do bloco de código do `while` como feito abaixo.

In [ ]:
```
senha_estah_errada = True
while senha_estah_errada:
    senha = int(input("Digite a sua senha numérica: "))
    if senha == 1234:
        senha_estah_errada = False
    else:
        print("SENHA INCORRETA")
print("SENHA CORRETA")
```

Faça você mesmo!

Modifique o código acima de modo que ele permita no máximo 3 tentativas para acertar a senha.

Dica: use uma variável para contar quantas tentativas foram feitas!

Mais exemplos

Os divisores de um número n são todos os números entre 1 e n que dividem n de forma exata (sem resto).

Para descobrir quais são os divisores de n, portanto, precisamos de um laço que, a cada iteração, assuma um valor diferente entre 1 e n e teste se esse valor é divisor de n.

In [15]:
```
n = int(input("Digite um valor para n: "))
for i in range(1,n+1):
    if n % i == 0:
        print(i, "é um divisor de", n)
```

```
Digite um valor para n: 11.1

---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-15-c6a3ca9c4862> in <module>()
----> 1 n = int(input("Digite um valor para n: "))
      2 for i in range(1,n+1):
      3     if n % i == 0:
      4         print(i, "é um divisor de", n)

ValueError: invalid literal for int() with base 10: '11.1'
```

Outro problema interessante é descobrir qual é o maior elemento dentre vários que foram digitados pelo usuário.

Na aula anterior vimos como fazer isso quando o usuário digita 2 ou 3 números: com `if..elif..else` e algumas comparações já tínhamos a resposta.

Mas quando o usuário entra com um valor desconhecido de números (n números), o problema fica mais complicado, pois mesmo que você soubesse qual é o valor de n, as comparações seriam muitas!

Uma boa ideia para resolver esse problema é a seguinte: vamos manter qual é o maior número dentre os números já lidos. Por exemplo, se o usuário já digitou a sequência de números $3, 7, 2, 6, -3, 8, 1, 4$, então o maior lido até o momento é o 8. Quando o usuário digitar um novo número x, temos apenas duas possibilidades: x é maior do que 8 ou não, isto é, o novo número lido pode ser ou não maior do que o maior número que foi lido até o momento. Se $x > 8$, então certamente x é maior do que qualquer outro número na sequência (não é necessário comparar x com todos os números da sequência para decidir isso). Note então que basta duas comparações para corretamente manter o maior número.

Veja a solução a seguir.

In [18]:
```
n = int(input("Digite a quantidade de números que serão fornecidos: "))
# não sabemos quais números serão fornecidos (podem ser negativos ou positivos)
# então vamos assumir que o primeiro desses números é o maior
maior = int(input("Digite um novo número: "))

# o laço a seguir se repete n-1 vezes pois já lemos um dos números
# note como o valor que i assume em cada iteração não é importante
for i in range(n):
    valor = int(input("Digite um novo número: "))
    # esse novo valor que foi fornecido pode ser maior do que o que estávamos considerando como maior antes
    # então atualizamos o valor da variável para armazenar corretamente o maior de todos
    if valor > maior:
        maior = valor

# apenas depois que todos os n números foram fornecidos é que temos certeza de qual é o maior
# por isso esse comando está fora do bloco do laço for
print("O maior valor fornecido foi", maior)
```

```
Digite a quantidade de números que serão fornecidos: 5
Digite um novo número: 10
Digite um novo número: 2
Digite um novo número: 3
Digite um novo número: 4
Digite um novo número: 1
Digite um novo número: 6
O maior valor fornecido foi 10
```

O logaritmo de um número é o expoente a que outro valor fixo, a base, deve ser elevado para produzir este número. Por exemplo, o logaritmo de 1000 na base 10 é 3 porque 10 ao cubo é 1000 ($1000 = 10 \times 10 \times 10 = 10^3$). De maneira geral, para quaisquer dois números reais b e x, onde b é positivo e $b \neq 1$,

$$y = b^x \Leftrightarrow x = \log_b(y).$$

Uma definição equivalente é a seguinte. O logaritmo de um número y na base b é no máximo a quantidade de vezes que y pode ser dividido por b até atingir um valor menor do que 1.

Por exemplo, o logaritmo de 9 na base 2 é no máximo 4, pois $9/2 = 4.5$, $4.5/2 = 2.25$, $2.25/2 = 1.125$ e $1.125/2 = 0.5625$.

In [ ]:
```
y = int(input("Digite o número para o qual deseja calcular o logaritmo: "))
b = int(input("Digite a base do logaritmo: "))

log = 0
while y > 1:
    y = y/b
    log = log + 1

print("O logaritmo é no máximo", log)
```

Note que no código anterior o valor de `y` era constantemente atualizado, de forma que nós "perdemos" o valor inicial.

Caso a mensagem final precisasse mostrar esse valor, não seria possível. Podemos corrigir isso usando uma nova variável para guardar o valor inicial de `y`.

In [ ]:
```
y = int(input("Digite o número para o qual deseja calcular o logaritmo: "))
b = int(input("Digite a base do logaritmo: "))

auxy = y
log = 0
while y > 1:
    y = y/b
    log = log + 1

print("O logaritmo de", auxy, "na base", b, "é no máximo", log)
```

Suponha agora que queremos encontrar todas soluções para a equação $x_1 + x_2 = C$, onde C é algum valor fornecido pelo usuário.

Vamos assumir que $x_1 \geq 0$ e $x_2 \geq 0$ são ambas variáveis inteiras.

Por exemplo, se $C = 3$, então todas as soluções possíveis são:

x_1	x_2
0	3
1	2
2	1
3	0

Note como o valor máximo que tanto x_1 quanto x_2 podem assumir é C.

In [ ]:
```
C = int(input("Digite o valor de C: "))
for x1 in range(C+1): # x1 pode assumir valores entre 0 e C
    # para cada valor que x1 assume, devemos testar todos os valores possíveis de x2
    for x2 in range(C+1): # x2 também pode assumir valores entre 0 e C
        # agora que temos um valor para x1 e outro para x2, verificamos se eles somam C
        if x1 + x2 == C:
            print("Solução possível: x1 =", x1, "e x2 = ", x2)
```

E se a equação fosse $x_1 + x_2 + x_3 = C$?

In [ ]:
```
C = int(input("Digite o valor de C: "))
for x1 in range(C+1):
    for x2 in range(C+1):
        for x3 in range(C+1):
            if x1 + x2 + x3 == C:
                print("Solução possível: x1 = ", x1, ", x2 = ", x2, "e x3 = ", x3)
```

Note que, na equação $x_1 + x_2 = C$, quando temos um valor de x_1, então x_2 pode ser unicamente determinado: x_2 é $C - x_1$.

Similarmente, na equação $x_1 + x_2 + x_3 = C$, quando temos um valor de x_1 e um valor de x_2, então x_3 pode ser unicamente determinado: x_3 é $C - x_1 - x_2$.

Volte aos códigos acima e melhore-os levando em consideração essas novas informações.