Raciocínio Computacional

UNIDADE 03

Estruturas de repetição

Esta unidade tem por objetivo apresentar as estruturas de repetição. Comumente chamadas laços de repetição, elas são fundamentais no processo de codificação, uma vez que permitem executar blocos de comandos repetidos com muito menos código. Além disso, possibilitam automatizar tarefas de armazenamento em conjunto com estruturas de dados homogêneas e heterogêneas, objetos de estudo das próximas unidades.

Estruturas de repetição

As estruturas de repetição, também chamadas laços de repetição, possuem dois papéis importantes no desenvolvimento de *software*: reaproveitamento de código e automatização de tarefas.

Com o uso de estruturas de dados, pode-se repetir dez vezes o código de pedir um número e somar em uma variável. Com isso, tem-se muito menos código escrito fazendo a mesma função, o chamado reaproveitamento de código.

Em termos de automatização de tarefas, as estruturas de repetição permitem fazer interação com as estruturas de dados homogêneas e heterogêneas existentes no Python, as quais são locais de armazenamento, como variáveis, que possuem apenas um nome, porém permitindo acesso aos seus dados a partir de índices. Como não apresentam nomes diferentes, como seria o caso ao usar variáveis, essas estruturas de dados podem ser manipuladas dentro dos laços de repetição, uma vez que os índices de acesso podem ser gerados durante a interação.

Laços finitos

Em lógica clássica, existe uma estrutura de repetição dita finita, ou seja, que permite interagir um número finito de vezes sobre um bloco de código. Esse comportamento é realizado a partir do comando **para**, que é uma composição de um teste condicional sobre uma variável de controle e um bloco de ações, a qual será repetida o número de interações determinado por essa variável.

A sintaxe do comando **para** é:

```
para v de vi até vf passo p faça
    c1;
    c2;
    ...
fimpara;
```

Em que **v** é o nome da variável que vai assumir os valores da iteração; **vi** é o valor inicial da iteração; **vf** é o valor final da iteração; **p** é o incremento com que os valores são iterados; **c1, c2, ...** são os comandos a ser executados dentro do laço.

Exemplo de aplicação 1: Elabore um programa que solicite ao usuário dez números e efetue a soma, exibindo o resultado.

```
1. início
2.     inteiro: num, soma, contador;
3.     soma <- 0;
```

```
4.     para contador de 1 até 10 passo 1 faça
5.         escreva ("Digite um número para somar: ");
6.         leia (num);
7.         soma <- soma + num;
8.     fimpara
9.     escreva ("A soma dos números é ", soma);
10. fim.
```

Neste exemplo, é possível perceber como ocorre o reaproveitamento de código. Caso ele fosse feito sem o uso da estrutura de repetição, as linhas 05, 06 e 07 teriam de ser repetidas dez vezes, o que levaria a um código muito maior.

Como ficaria nossa implementação em Python?

#ParaTodosVerem

```
1 num = 0
2 soma = 0
3 contador = 0
4
5 for contador in range (0,10): # fará de 0 zero até 9 nove
6     print ("Digite um número para somar: ");
7     num = int(input())
8     soma = soma + num
9 print ("A soma dos números é ", soma);
```

É importante destacar que a variável de controle pode ter seu valor utilizado dentro do laço. O mesmo exemplo poderia ser reescrito com a seguinte modificação na linha 05:

```
5. escreva ("Digite o ", contador , "° número para somar: ");
```

Assim, cada frase ficaria específica de acordo com a iteração do laço.

Exemplo de aplicação 2: Elabore um programa que calcule o fatorial de um número. Exemplo: Fatorial de 5 é igual a 5*4*3*2*1.

```
1. início
2.     ingeiro: num, fatorial, contador;
3.     fatorial <- 1;
4.     escreva ("Digite um número: ");
5.     leia (num);
6.     para contador de num até 1 passo -1 faça
7.         fatorial <- fatorial * contador;
8.     fimpara
9.     escreva ("O fatorial de ", num, " é ", fatorial);
```

```
10. fim.
```

Neste exemplo, é fundamental o uso da variável contadora no cálculo. Para facilitar a leitura, de acordo com a definição de fatorial, faz-se a contagem invertida, começando o laço em **num** e indo até 1, para ser a forma mais lógica de executar a tarefa. Contudo, para que isso seja possível, é necessário mudar o passo para -1.

Agora, veja esta lógica em Python. Bacana, não?

#ParaTodosVerem

```
1  def main():
2      n = int(input("Digite o valor de n: "))
3      fat = 1
4      i = 2
5      while i <= n:
6          fat = fat*i
7          i = i + 1
8
9      print("O valor de %d! eh =" %n, fat)
10
11 main()
```

Laço finito – implementação em Python

O comando em Python que implementa a estrutura de repetição finita é o comando ***for***, cuja sintaxe é:

```
for var in <<sequência>>:
    <<bloco de ações>>
```

Em Python, o comando ***for*** atua, diferentemente de outras linguagens de programação, como um iterador para determinada sequência, que, neste caso, pode ser qualquer estrutura do Python, como, por exemplo, uma *string*, um *range* finito, uma estrutura de dados (homogênea ou heterogênea), entre outras.

SAIBA MAIS

Veja outros exemplos de ***for*** no W3School.

W3School não é qualquer site, mas o site acadêmico do W3Consorsium. Os principais desenvolvedores em algum momento no mundo se uniram para discutir padrões e formaram

um grupo, o W3Consorcium. O W3School é sua referência acadêmica, ou seja, não estamos indicando qualquer site de código, mas O principal deles, internacionalmente falando.

Exemplo de aplicação 3: Elabore um programa que solicite ao usuário seu nome e mostre cada uma das letras do nome em uma linha diferente do console.

```
1. nome = input("Por favor, digite o seu nome: ")
2. for letra in nome:
3.     print(letra)
```

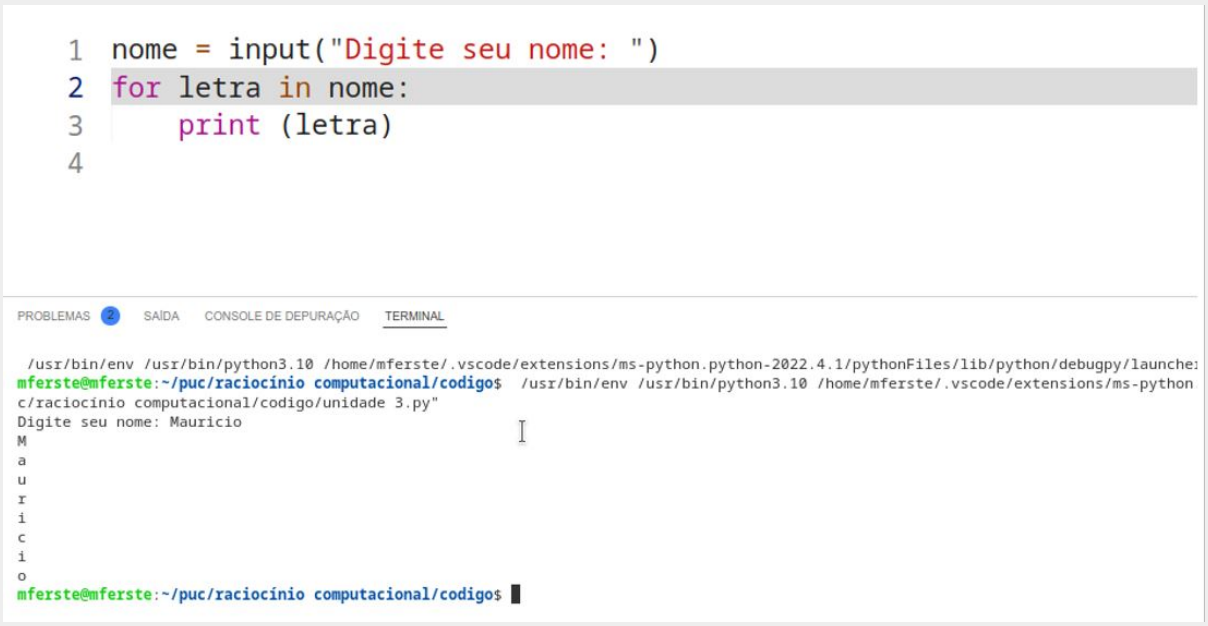
C:\

```
/Users/user/untitled1/bin/python
/Users/user/projects/untitled1/ExAplic03.py
Por favor, digite o seu nome: Joaquim
J
o
a
q
u
i
m

Process finished with exit code 0
```

Neste exemplo, a sequência utilizada é uma *string*. O laço ***for*** faz iteração com a *string*, passando por cada uma de suas letras.

Veja, agora, como fica a saída utilizando Python:

#ParaTodosVerem



Exemplo de aplicação 4: Elabore um programa que solicite três números, some-os e exiba o resultado para o usuário.

```
1. soma = 0
2. for i in range(3):
3.     num = int(input("Digite o " + str(i + 1) + " número: "))
4.     soma += num
5. print("A soma dos números é", soma)
```

C:\

```
/Users/user/untitled1/bin/python
/Users/user/projects/untitled1/ExAplic04.py
Digite o 1 número: 10
Digite o 2 número: 20
Digite o 3 número: 30
A soma dos números é 60


Process finished with exit code 0
```

Neste exemplo, a sequência utilizada é um *range* finito. No caso, **range(3)** retorna valores de 0 a 2, os quais são atribuídos à variável contador **i**. Em função disso, na linha 03, a frase é composta com **i + 1**, para gerar o número correto. Nessa mesma linha, é possível perceber a necessidade de conversão do valor de **i + 1** para texto, a fim de somá-lo às demais *strings* da frase. Isso ocorre porque o Python é fortemente tipado, conforme mencionado na Unidade de Estudo 1.

Como desenvolveria em Python? Fica aqui um desafio!

Tipo range

O tipo *range* retorna uma sequência imutável de valores numéricos entre um valor mínimo e um valor máximo, podendo definir um passo de iteração. Possui duas sintaxes de uso, como segue:

```
range(limite_final)
range(valor_inicial, limite_final[, passo])
```

Na primeira sintaxe, **limite_final** define o valor final do *range* (excluído este valor), com valor inicial 0. Por exemplo:

```
range(5) => [0, 1, 2, 3, 4]
```

Esta forma de utilização pode ser vista no exemplo de aplicação 4.

Na segunda sintaxe, **valor_inicial** indica o primeiro valor do *range* (inclusive este valor), com **limite_final** definindo o valor final (excluído este valor), com o incremento sendo definido pelo **passo**, que é opcional, com um valor-padrão igual a 1. Por exemplo:

```
range(3, 8) => [3, 4, 5, 6, 7]
range(2, 9, 2) => [2, 4, 6, 8]
range(7, 1, -1) => [7, 6, 5, 4, 3, 2]
```

Exemplo de aplicação 5: Elabore um programa que calcule o fatorial de um número, exibindo a informação de como é feito o cálculo. Exemplo: Fatorial de 5 é igual a 5*4*3*2*1.

```
1. fatorial = 1
2. expressao = "Expressão: "
3. num = int(input("Digite um número para o cálculo do
   fatorial: "))
4. for i in range(num, 0, -1):
5.     fatorial *= i
```

```
6.      expressao += str(i)
7.      if i > 1:
8.          expressao += " * "
9. print("O valor do fatorial de", num, "é", fatorial,
   expressao)
```

C:\

```
/Users/user/untitled1/bin/python
/Users/user/projects/untitled1/ExAplic05.py
Digite um número para o cálculo do fatorial: 5
O valor do fatorial de 5 é 120 Expressão: 5 * 4 * 3 * 2 *
1


Process finished with exit code 0
```

Controle de fluxo

O Python apresenta as seguintes estruturas de controle de fluxo:

Controle de fluxo

Comando	Descrição
break	Interrompe o fluxo da estrutura de repetição.
continue	Força uma nova iteração a partir do ponto corrente.
pass	Preenche estrutura vazia, não permitida em Python.

O comando ***break*** interrompe um laço no ponto onde estiver do código, passando a execução do programa para a linha imediatamente posterior ao final do laço.

Exemplo de aplicação 6: Elabore um programa que solicite ao usuário dez números e efetue a soma, exibindo o resultado. Contudo, se em algum momento o número digitado for 0, deve interromper o laço, mostrando a soma apenas dos valores

informados até então.

```
1. soma = 0
2. for i in range(10):
3.     num = int(input("Digite o " + str(i + 1) + " número:
   "))
4.     if num == 0:
5.         break
6.     soma += num
7. print("A soma dos números é", soma)
```

C:\

```
/Users/user/untitled1/bin/python
/Users/user/projects/untitled1/ExAplic06.py
Digite o 1 número: 10
Digite o 2 número: 20
Digite o 3 número: 30
Digite o 4 número: 0
A soma dos números é 60


Process finished with exit code 0
```

O comando ***continue*** força uma nova iteração a partir do ponto em que o programa se encontra, não sendo executado o resto do laço.

Exemplo de aplicação 7: Elabore um programa que solicite ao usuário dez números e efetue a multiplicação, exibindo o resultado. No entanto, se em algum momento o número digitado for 0, deve pular esta iteração, para que o valor não seja zerado.

```
1. mult = 1
2. for i in range(10):
3.     num = int(input("Digite o " + str(i + 1) + " número:
   "))
4.     if num == 0:
5.         continue
6.     mult *= num
7. print("A multiplicação dos números é", mult)
```

C:\

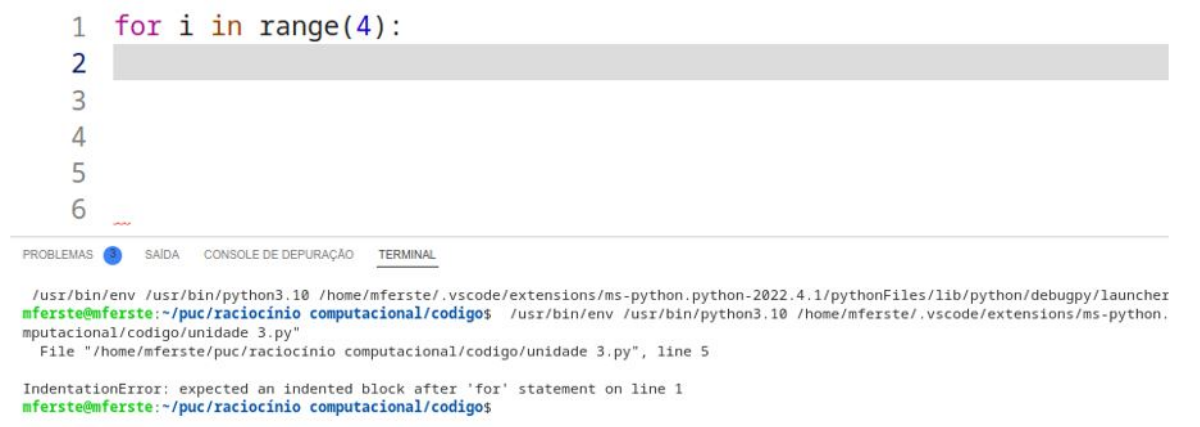
```
/Users/user/untitled1/bin/python
/Users/user/projects/untitled1/ExAplic07.py
Digite o 1 número: 10
Digite o 2 número: 5
Digite o 3 número: 3
Digite o 4 número: 0
Digite o 5 número: 0
Digite o 6 número: 0
Digite o 7 número: 1
Digite o 8 número: 1
Digite o 9 número: 1
Digite o 10 número: 1
A multiplicação dos números é 150


Process finished with exit code 0
```

O comando ***pass*** preenche uma estrutura que, porventura, tenha de ficar vazia. Normalmente, é utilizado para testar códigos ainda não terminados.

```
1. for i in range(4):
2.     pass
```

Veja, por exemplo, sem o comando pass:

#ParaTodosVerem

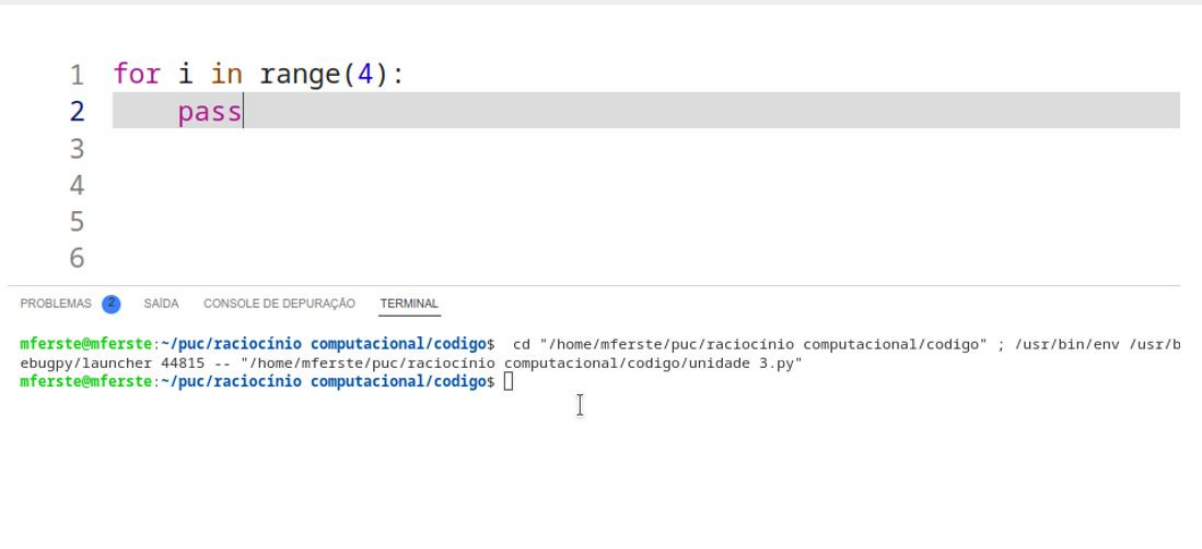


Ocorreu um erro, que se deve à falta de código.

Vale ressaltar que o comando *pass* pode ser utilizado com qualquer estrutura de controle do Python, a fim de que ela não gere erro por estar vazia.

Agora com o comando pass:

#ParaTodosVerem



Não tem erro, correto?

Laços infinitos

Enquanto os laços finitos têm um número fixo de iterações, os laços infinitos executam de acordo com uma determinada condição, a qual gera saída do laço conforme sua execução interna, seja por um comando de controle de fluxo, seja pela alteração do valor da variável analisada na condição do laço.

Em lógica clássica, existem alguns tipos de laço infinito, porém o Python só utiliza laços com teste no início. Sendo assim, aqui será vista apenas a estrutura **enquanto**, cuja sintaxe é:

```
enquanto <<condição>> faça:
    c1;
    c2;
    ...
fimenquanto;
```

Em que onde **<<condição>>** é o teste condicional efetuado; **c1, c2, ...** são os comandos a ser executados dentro do laço.

Dica: teremos de utilizar algum laço como o mostrado no nosso menu principal, do qual somente sairemos quando atingirmos uma condição de variável, como, por exemplo, enquanto não digitar, algo não vai finalizar o programa.

Exemplo de aplicação 8: Elabore um programa que solicite ao usuário que digite indefinidamente números e efetue a soma, parando apenas quando o usuário digitar o número 0.

```
1. início
2.     inteiro: num, soma;
3.     soma <- 0;
4.     num <- -1;
5.     enquanto (num <> 0) faça
6.         escreva ("Digite um número para somar (0 finaliza):
   ");
7.         leia (num);
8.         soma <- soma + num;
9.     fimenquanto
10.    escreva ("A soma dos números é ", soma);
11. fim.
```

Laço infinito – implementação em Python

O comando em Python que implementa a estrutura de repetição infinita é o comando ***while***, cuja sintaxe é:

```
while <<condição>>:
    <<bloco de ações>>
```

Exemplo de aplicação 9: Elabore um programa que solicite ao usuário que digite indefinidamente números e efetue a soma, parando apenas quando o usuário digitar o número 0.

```
1. soma = 0
2. num = -1
3. while num != 0:
4.     num = int(input("Digite um número para somar (0
   finaliza): "))
5.     soma += num
6. print("A soma dos números é ", soma)
```

C:\

```
/Users/user/untitled1/bin/python
/Users/user/projects/untitled1/ExAplic09.py
Digite um número para somar (0 finaliza): 10
Digite um número para somar (0 finaliza): 20
Digite um número para somar (0 finaliza): 30
Digite um número para somar (0 finaliza): 0
A soma dos números é 60


Process finished with exit code 0
```

Este mesmo exemplo pode ser implementado fazendo uso de uma condição infinita, com o comando de controle de fluxo ***break***.

```
1. soma = 0
2. num = -1
3. while True:
4.     num = int(input("Digite um número para somar (0
   finaliza): "))
5.     if num == 0:
6.         break;
7.     soma += num
8. print("A soma dos números é ", soma)
```

C:\

```
Users/user/untitled1/bin/python
/Users/user/projects/untitled1/ExAplic09.py
Digite um número para somar (0 finaliza): 10
Digite um número para somar (0 finaliza): 20
Digite um número para somar (0 finaliza): 30
Digite um número para somar (0 finaliza): 0
A soma dos números é 60


Process finished with exit code 0
```

Da mesma forma, podem ser usados os comandos ***continue*** e ***pass*** com o comando ***while***.

Validação de dados com tratamento de exceções

Até o momento, os códigos apresentados levam em consideração que o usuário está fazendo a entrada dos dados de forma correta, sem qualquer validação. No caso de, por exemplo, digitar um texto em uma entrada com conversão para inteiro, o resultado será:

```
num = int(input("Digite um número para somar (0 finaliza): "))
```

C:\

```
/Users/user/untitled1/bin/python
/Users/user/projects/untitled1/First.py
Digite um número para somar (0 finaliza): zero
Traceback (most recent call last):
  File "/Users/user/projects/untitled1/First.py", line 2,
in <module>
    num = int(input("Digite um número para somar (0
finaliza): "))
ValueError: invalid literal for int() with base 10: 'zero'


Process finished with exit code 1
```

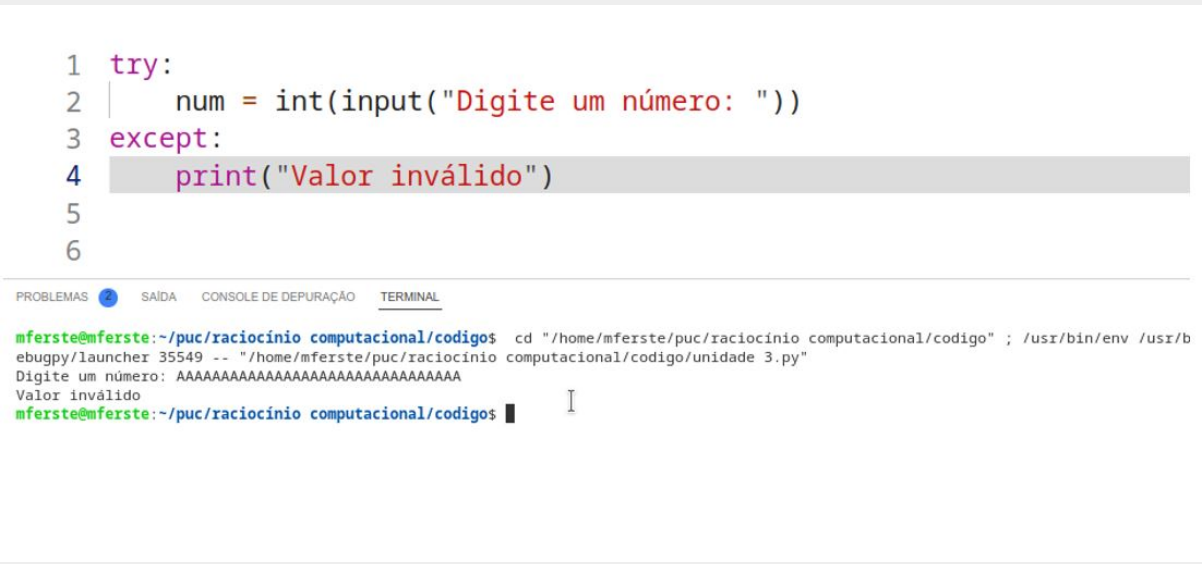
Para evitar esse tipo de erro, ele deve ser capturado por um sistema de tratamento de exceção, feito pelo comando ***try...except***.

```
1. try:
2.     num = int(input("Digite um número: "))
3. except:
4.     print("Valor inválido")
```

Neste caso, em vez de disparar a exceção, ela é capturada pela cláusula ***except***, mostrando a mensagem de valor inválido. Essa cláusula captura todo tipo de erro, dando um tratamento comum independentemente de qual seja.

Veja a execução a seguir:

#ParaTodosVerem



Para capturar erros específicos, deve-se informar o tipo. No exemplo anterior, é possível verificar que o erro é do tipo ***ValueError***; assim:

```
try:
    num = int(input("Digite um número: "))
except ValueError:
    print("Valor inválido")
except:
    print("Outro tipo de erro ocorreu!")
```

Neste caso, podem ser identificados vários tipos de erro, tratando cada um de forma individual.

Para validar uma entrada de dados, não basta indicar que houve um erro, mas dar a oportunidade de entrar os dados de forma correta novamente.

Exemplo de aplicação 10: Elabore um programa que solicite um número inteiro ao usuário, validando e imprimindo esse número.

```
while True:
    try:
        num = int(input("Digite um número: "))
        break
    except ValueError:
        print("Valor inválido")
print("Número validado:", num)
```

C:\

```
/Users/user/untitled1/bin/python
/Users/user/projects/untitled1/ExAplic10.py
Digite um número: zero
Valor inválido
Digite um número: 3.4
Valor inválido
Digite um número: 3
Número validado: 3


Process finished with exit code 0
```

É importante perceber neste exemplo que, mesmo entrando um formato válido de número, como no caso de 3.4, a exceção é lançada, uma vez que está tentando converter a entrada de dados para um valor do tipo inteiro.

Exemplo de aplicação 11: Elabore um programa que solicite um número decimal ao usuário, validando e imprimindo esse número.

```
1. while True:
2.     try:
3.         num = float(input("Digite um número decimal: "))
4.         break
5.     except ValueError:
6.         print("Valor inválido")
7. print("Número validado:", num)
```

C:\

```
/Users/user/untitled1/bin/python
/Users/user/projects/untitled1/ExAplic11.py
Digite um número decimal: 3,2
Valor inválido
Digite um número decimal: 3.2
Número validado: 3.2


Process finished with exit code 0
```

Além da validação pelo tipo de dado correto, é possível validar valores de acordo com determinado critério. Neste caso, a saída do laço só deve ser chamada após o critério ser satisfeito.

Exemplo de aplicação 12: Elabore um programa que solicite um número inteiro ao usuário, em um intervalo entre 1 e 5.

```
1. while True:
2.     try:
3.         num = int(input("Digite um número inteiro entre 1 e
   5: "))
4.         if 1 <= num <= 5:
5.             break
6.         else:
7.             print("O número deve estar entre 1 e 5.")
8.     except ValueError:
9.         print("Valor inválido")
10. print("Número validado:", num)
```

C:\

```
/Users/user/untitled1/bin/python
/Users/user/projects/untitled1/ExAplic12.py
Digite um número inteiro entre 1 e 5: 3.1
Valor inválido
Digite um número inteiro entre 1 e 5: 7
O número deve estar entre 1 e 5.
Digite um número inteiro entre 1 e 5: 3
Número validado: 3


Process finished with exit code 0
```

EXERCÍCIO

Laços finitos

Exercício de fixação 1: Crie um programa que solicite ao usuário dez números, contando quantos são pares e quantos são ímpares e informando ao final essas informações.

Resolução do exercício

Resolução exercício 1

#ParaTodosVerem

```
1  contPar = 0;
2  contImpar = 0;
3  for i in range(10):
4      num = int(input("Digite um número: "))
5      if num % 2 == 0:
6          contPar += 1
7      else:
8          contImpar += 1
9  print("Dos 10 números", contPar, "são pares e", contImpar, "são ímpares.")
10
```

```
PROBLEMAS  SAÍDA  CONSOLE DE DEPURAÇÃO  TERMINAL

mferste@mferste:~/puc/raciocínio computacional/codigos$ cd "/home/mferste/puc/raciocínio computacional/codigo" ; /usr/bin/env /usr/bin/python3.10 /home/mf
ebugpy/launcher 45785 -- "/home/mferste/puc/raciocínio computacional/codigo/unidade 3.py"
Digite um número: 0
Digite um número: 1
Digite um número: 2
Digite um número: 3
Digite um número: 4
Digite um número: 5
Digite um número: 6
Digite um número: 7
Digite um número: 8
Digite um número: 9
Dos 10 números 5 são pares e 5 são ímpares.
mferste@mferste:~/puc/raciocínio computacional/codigos
```

C:\

```
/Users/user/untitled1/bin/python
/Users/user/projects/untitled1/ExFix01.py
Digite um número: 1
Digite um número: 2
Digite um número: 3
Digite um número: 4
Digite um número: 5
Digite um número: 6
Digite um número: 7
Digite um número: 8
Digite um número: 9
Digite um número: 10
Dos 10 números 5 são pares e 5 são
ímpares.


Process finished with exit code 0
```

Exercício de fixação 2: Crie um programa que peça ao usuário cinco números e informe qual é o menor e qual é o maior deles.

 Resolução do exercício 

Resolução exercício 2

```
menor = 1000000
maior = -1000000
for i in range(5):
    num = int(input("Digite um número: "))
    if num < menor:
        menor = num
    if num > maior:
        maior = num
print("O menor número é", menor, "e o maior
número é", maior)
```

C:\

```
Users/user/untitled1/bin/python
/Users/user/projects/untitled1/ExFix02.py
Digite um número: 5
Digite um número: 9
Digite um número: 1
Digite um número: 3
Digite um número: 8
O menor número é 1 e o maior número é 9


Process finished with exit code 0
```

Exercício de fixação 3: Crie um programa que receba um texto digitado pelo usuário e o imprima apenas com consoantes, removendo as vogais. Observação: desconsiderar letras maiúsculas e acentos.

 Resolução do exercício 

Resolução exercício 3

```
  1  frase = input("Digite um texto
qualquer:")
  2  consoantes = ""
  3  for letra in frase:
  4      if letra.lower() not in "aeiou":
  5          consoantes = consoantes + letra
  6
  7  print(consoantes)
```

C:\

```
/Users/user/untitled1/bin/python
/Users/user/projects/untitled1/ExFix03.py
Digite um texto qualquer: Minha terra tem
palmeiras onde canta o sabiá
Mnh trr tm plmrs nd cnt  sbá


Process finished with exit code 0
```

Exercício de fixação 4: Crie um programa que solicite um número ao usuário e exiba a tabuada dele de 1 a 10, no formato:

Tabuada do n

n x 1 = n

n x 2 = 2n

...

n x 10 = 10n

 Resolução do exercício 

Resolução exercício 4

```
1. num = int(input("Informe um número: "))
2. print("Tabuada do", num)
3. for i in range(1, 11):
4.     mult = i * num
5.     print(i, "x", num, "=", mult)
```

C:\

```
/Users/user/untitled1/bin/python
/Users/user/projects/untitled1/ExFix04.py
Informe um número: 7
Tabuada do 7
1 x 7 = 7
2 x 7 = 14
3 x 7 = 21
4 x 7 = 28
5 x 7 = 35
6 x 7 = 42
7 x 7 = 49
8 x 7 = 56
9 x 7 = 63
10 x 7 = 70


Process finished with exit code 0
```

Exercício de fixação 5: Crie um programa que peça dois números ao usuário – o primeiro será o numerador e o segundo, o expoente. A saída do programa deve ser o resultado da operação: numerador elevado ao expoente. Observação: não usar função interna que calcula automaticamente potência.

 Resolução do exercício 

Resolução exercício 5

```
1. numerador = int(input("Digite o valor do
   numerador: "))
2. expoente = int(input("Digite o valor do
   expoente: "))
3. result = 1
4. for i in range(expoente):
5.     result *= numerador
6. print(numerador, "elevado a", expoente, "é
   igual a", result)
```

C:\

```
/Users/user/untitled1/bin/python
/Users/user/projects/untitled1/ExFix05.py
Digite o valor do numerador: 2
Digite o valor do expoente: 5
2 elevado a 5 é igual a 32


Process finished with exit code 0
```

Laços infinitos

Exercício de fixação 6: Crie um programa que peça para o usuário inserir um *login* e uma senha. Caso os valores sejam iguais, informar que os dados são inválidos e pedir novamente as informações. Caso contrário, exibir a mensagem "Bem-vindo ao sistema!".

 Resolução do exercício 

Resolução exercício 6

```
7.  while(True):
8.      login = input("Digite um login: ")
9.      senha = input("Digite uma senha: ")
10.      if login == senha:
11.          print("Dados inválidos. Por
favor, redigite.")
12.      else:
13.          break
14.  print("Bem-vindo ao sistema!")
```

C:\

```
/Users/user/untitled1/bin/python
/Users/user/projects/untitled1/ExFix06.py
Digite um login: senha
Digite uma senha: senha
Dados inválidos. Por favor, redigite.
Digite um login: login
Digite uma senha: senha
Bem-vindo ao sistema!


Process finished with exit code 0
```

Exercício de fixação 7: Crie um programa que solicite ao usuário vários números e, ao digitar 0, calcule a média dos números informados.

 Resolução do exercício 

Resolução exercício 7

```
1. contador = 0
2. soma = 0
3. while True:
4.     num = int(input("Digite um número (0 para
   encerrar): "))
5.     if num == 0:
6.         break
7.     else:
8.         contador += 1
9.         soma += num
10. if contador == 0:
11.     print("Não foram inseridos números a
    calcular.")
12. else:
13.     media = soma / contador
14.     print("A média dos", contador, "números
    é", media)
```

C:\

```
/Users/user/untitled1/bin/python
/Users/user/projects/untitled1/ExFix07.py
Digite um número (0 para encerrar): 5
Digite um número (0 para encerrar): 8
Digite um número (0 para encerrar): 9
Digite um número (0 para encerrar): 8
Digite um número (0 para encerrar): 2
Digite um número (0 para encerrar): 2
Digite um número (0 para encerrar): 0
A média dos 6 números é 5.666666666666667


Process finished with exit code 0
```

Exercício de fixação 8: Crie um programa que gere a série de Fibonacci enquanto o valor for menor que um valor informado pelo usuário. Observação: série de Fibonacci = 0, 1, 1, 2, 3, 5, 8 ,13, 21, 34, 55, ... é formada por 0, 1 e, partir deste ponto, sempre será a soma dos dois valores anteriores.

 Resolução do exercício 

Resolução exercício 8

```
 1. valor = 0;
 2. max = int(input("Informe o valor-limite: "))
 3. num1 = 0
 4. num2 = 1
 5. serie = "0, 1"
 6. while valor < max:
 7.     valor = num1 + num2
 8.     if valor < max:
 9.         num1 = num2
10.         num2 = valor;
11.         serie += ", " + str(valor)
12. print("Fibonacci:", serie)
```

C:\

```
/Users/user/untitled1/bin/python
/Users/user/projects/untitled1/ExFix08.py
Informe o valor-limite: 300
Fibonacci: 0, 1, 1, 2, 3, 5, 8, 13, 21,
34, 55, 89, 144, 233


Process finished with exit code 0
```

Desafio

Exercício de fixação 9: Crie um programa que simule um carrinho de compras, solicitando o nome do produto (não pode ser vazio), seu valor (valor decimal, positivo) e quantidade a ser comprada (valor inteiro, positivo). Ao incluir um produto, deve perguntar se o usuário deseja fechar o pedido ou incluir mais produtos. Todos os dados devem ser validados. Ao final da compra, deve ser informado o valor total do pedido.

 Resolução do exercício 

Resolução exercício 9

```
1. print("Sistema de compras")
2. total = 0
3. continuar = True
4. while continuar:
5.     while True:
6.         produto = input("Informe o nome do
   produto a ser comprado: ")
7.         if produto != "":
8.             break
9.         else:
10.            print("Nome do produto
   inválido.")
11.    while True:
12.        try:
```

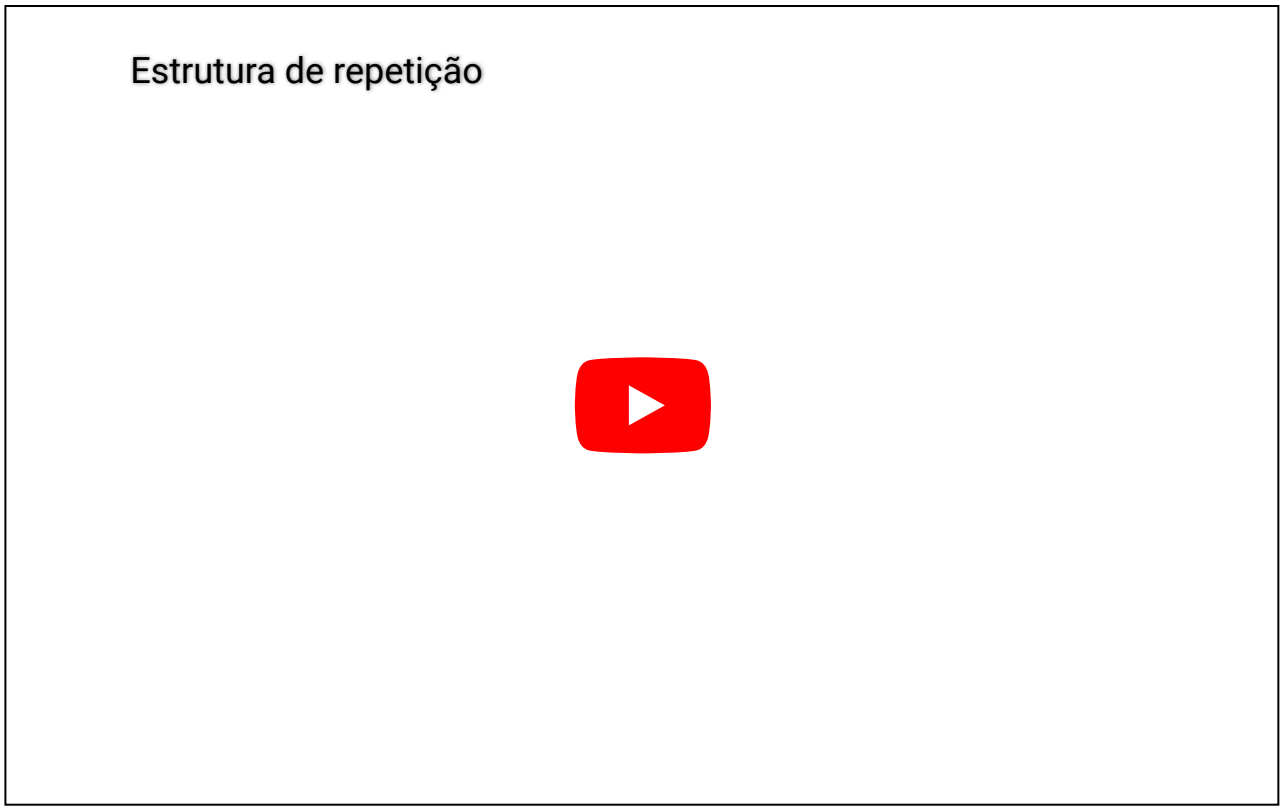
```
            valor = float(input("Informe o valor unitário do produto: "))
            if valor <= 0:
                print("O valor do produto deve ser maior que 0.")
            else:
                break
        except ValueError:
            print("Valor do produto inválido.")
    while True:
        try:
            quant = int(input("Informe a quantidade do produto a ser comprada: "))
            if valor <= 0:
                print("O valor do produto deve ser maior que 0.")
            else:
                break
        except ValueError:
            print("Valor do produto inválido.")

    total += quant * valor

    resp = input("Produto inserido com sucesso. Deseja comprar mais algum produto (s/n)? ")
    if resp == "n":
        continuar = False
print(f"Valor total da compra: R$ {total:.2f}")
```

C:\

```
/Users/user/untitled1/bin/python
/Users/user/projects/untitled1/ExFix09.py
Sistema de compras
Informe o nome do produto a ser comprado:
Nome do produto inválido.
Informe o nome do produto a ser comprado:
Caneta
Informe o valor unitário do produto: 8
Informe a quantidade do produto a ser
comprada: 3
Produto inserido com sucesso. Deseja
comprar mais algum produto (s/n)? s
Informe o nome do produto a ser comprado:
Estojo
Informe o valor unitário do produto: 33,50
Valor do produto inválido.
Informe o valor unitário do produto: 33.50
Informe a quantidade do produto a ser
comprada: 2
Produto inserido com sucesso. Deseja
comprar mais algum produto (s/n)? n
Valor total da compra: R$ 91.00


Process finished with exit code 0
```

Estrutura de repetição

Neste vídeo, veremos como utilizar estruturas de repetição e controles de fluxo.



Conclusão

A respeito do uso de estruturas de repetição na programação em Python, podemos concluir que:

- As estruturas de repetição podem ser finitas ou infinitas.
- Estruturas de repetição finitas possuem limites inferior e superior para suas iterações.
- Estruturas de repetição finitas iteram sobre sequências, como *range*, *strings*, entre outras.
- Estruturas de repetição infinitas fazem a análise de uma condição de parada.
- Os laços podem alterar seu fluxo de controle pelas cláusulas *break* e *continue*.
- Os laços podem utilizar a cláusula *pass* para não precisar passar por um bloco de ações.
- A partir da captura de exceções com um bloco *try...except*, é possível interceptar erros.
- Pela interceptação de erros, é possível efetuar validação de dados informados pelo usuário.

Referências

BANIN, S. L. **Python 3**: conceitos e aplicações uma abordagem didática. São Paulo: Érica, 2018.

FORBELLONE, A. L. V.; EBERSPÄCHER, H. F. **Lógica de programação**: a construção de algoritmos e estruturas de dados. São Paulo: Prentice Hall, 2005.

PYTHON SOFTWARE FOUNDATION. **Documentação Python 3.9.0**. Disponível em: https://docs.python.org/pt-br/3/. Acesso em: 27 out. 2020.