Curso livre de programação em Python

Encontro 6 - Tratamento de exceções

Prof. Louis Augusto
louis.augusto@ifsc.edu.br

Instituto Federal de Santa Catarina
Campus São José

Índice

Sumário

Exceções - introdução

Há dois erros principais quando se está programando, erro de sintaxe e erro de execução.

- Erro de sintaxe: ocorre quando cometemos uma falha típica da linguagem, como esquecer de colocar : depois da definição de um laço ou de uma expressão condicional.

```
# -*- coding:utf-8 -*-
for i in range(2)
    print("Teste {:2d}".format(i))
print("Linha 6")
```

Este programa não executa as impressões no interior do laço nem a impressão depois do laço. O programa é interrompido com o erro. Para corrigir basta inserir o caracter ':' no final da linha de definição do laço.

- Erro de execução: ocorre quando a digitação está correta, mas há algo proibido sendo executado, como a extrapolação do índice de um vetor, divisão por zero, operação de multiplicação usando strings etc.

Exceções - introdução

Há dois erros principais quando se está programando, erro de sintaxe e erro de execução.

- Erro de sintaxe: ocorre quando cometemos uma falha típica da linguagem, como esquecer de colocar : depois da definição de um laço ou de uma expressão condicional.

```
# -*- coding:utf-8 -*-
for i in range(2)
    print("Teste {:2d}".format(i))
print("Linha 6")
```

Este programa não executa as impressões no interior do laço nem a impressão depois do laço. O programa é interrompido com o erro. Para corrigir basta inserir o caracter ':' no final da linha de definição do laço.

- Erro de execução: ocorre quando a digitação está correta, mas há algo proibido sendo executado, como a extrapolação do índice de um vetor, divisão por zero, operação de multiplicação usando strings etc.

Exceções - introdução

Há dois erros principais quando se está programando, erro de sintaxe e erro de execução.

- Erro de sintaxe: ocorre quando cometemos uma falha típica da linguagem, como esquecer de colocar : depois da definição de um laço ou de uma expressão condicional.

```
# -*- coding:utf-8 -*-
for i in range(2)
    print("Teste {:2d}".format(i))
print("Linha 6")
```

Este programa não executa as impressões no interior do laço nem a impressão depois do laço. O programa é interrompido com o erro. Para corrigir basta inserir o caracter ':' no final da linha de definição do laço.

- Erro de execução: ocorre quando a digitação está correta, mas há algo proibido sendo executado, como a extrapolação do índice de um vetor, divisão por zero, operação de multiplicação usando strings etc.

Exceções - introdução

Há dois erros principais quando se está programando, erro de sintaxe e erro de execução.

- Erro de sintaxe: ocorre quando cometemos uma falha típica da linguagem, como esquecer de colocar : depois da definição de um laço ou de uma expressão condicional.

```
# -*- coding:utf-8 -*-
for i in range(2)
    print("Teste {:2d}".format(i))
print("Linha 6")
```

Este programa não executa as impressões no interior do laço nem a impressão depois do laço. O programa é interrompido com o erro. Para corrigir basta inserir o caracter ':' no final da linha de definição do laço.

- Erro de execução: ocorre quando a digitação está correta, mas há algo proibido sendo executado, como a extrapolação do índice de um vetor, divisão por zero, operação de multiplicação usando strings etc.

Exceções - introdução

Há dois erros principais quando se está programando, erro de sintaxe e erro de execução.

- Erro de sintaxe: ocorre quando cometemos uma falha típica da linguagem, como esquecer de colocar : depois da definição de um laço ou de uma expressão condicional.

```
# -*- coding:utf-8 -*-
for i in range(2)
    print("Teste {:2d}".format(i))
print("Linha 6")
```

  Este programa não executa as impressões no interior do laço nem a impressão depois do laço. O programa é interrompido com o erro. Para corrigir basta inserir o caracter ':' no final da linha de definição do laço.

- Erro de execução: ocorre quando a digitação está correta, mas há algo proibido sendo executado, como a extrapolação do índice de um vetor, divisão por zero, operação de multiplicação usando strings etc.

Exceções - introdução

Há dois erros principais quando se está programando, erro de sintaxe e erro de execução.

- Erro de sintaxe: ocorre quando cometemos uma falha típica da linguagem, como esquecer de colocar : depois da definição de um laço ou de uma expressão condicional.

```
# -*- coding:utf-8 -*-
for i in range(2)
    print("Teste {:2d}".format(i))
print("Linha 6")
```

  Este programa não executa as impressões no interior do laço nem a impressão depois do laço. O programa é interrompido com o erro. Para corrigir basta inserir o caracter ':' no final da linha de definição do laço.

- Erro de execução: ocorre quando a digitação está correta, mas há algo proibido sendo executado, como a extrapolação do índice de um vetor, divisão por zero, operação de multiplicação usando strings etc.

Exceções - introdução

Há dois erros principais quando se está programando, erro de sintaxe e erro de execução.

- Erro de sintaxe: ocorre quando cometemos uma falha típica da linguagem, como esquecer de colocar : depois da definição de um laço ou de uma expressão condicional.

```
# -*- coding:utf-8 -*-
for i in range(2)
    print("Teste {:2d}".format(i))
print("Linha 6")
```

  Este programa não executa as impressões no interior do laço nem a impressão depois do laço. O programa é interrompido com o erro. Para corrigir basta inserir o caracter ':' no final da linha de definição do laço.

- Erro de execução: ocorre quando a digitação está correta, mas há algo proibido sendo executado, como a extrapolação do índice de um vetor, divisão por zero, operação de multiplicação usando strings etc.

Exceções - introdução

Observe a execução deste programa:

```
mat = [None]*4
for i in range(5):
    mat[i] = i
print(mat)
```

Não há erro de sintaxe, o problema que ocorre é simplesmente que o vetor mat tem 4 posições, 0,1,2,3 e quis-se pôr um valor na posição mat[4] na última iteração do laço.

A resposta obtida é algo como:

```
Programas/prog1.py", line 13, in <module>
    mat[i] = i
IndexError: list assignment index out of range
```

Para que o programa não seja interrompido por erros de execução foram criadas as exceções.

Exceções - introdução

Observe a execução deste programa:

```
mat = [None]*4
for i in range(5):
    mat[i] = i
print(mat)
```

Não há erro de sintaxe, o problema que ocorre é simplesmente que o vetor mat tem 4 posições, 0,1,2,3 e quis-se pôr um valor na posição mat[4] na última iteração do laço.

A resposta obtida é algo como:

```
Programas/prog1.py", line 13, in <module>
    mat[i] = i
IndexError: list assignment index out of range
```

Para que o programa não seja interrompido por erros de execução foram criadas as exceções.

Exceções - introdução

Observe a execução deste programa:

```
mat = [None]*4
for i in range(5):
    mat[i] = i
print(mat)
```

Não há erro de sintaxe, o problema que ocorre é simplesmente que o vetor mat tem 4 posições, 0,1,2,3 e quis-se pôr um valor na posição mat[4] na última iteração do laço.

A resposta obtida é algo como:

```
Programas/prog1.py", line 13, in <module>
    mat[i] = i
IndexError: list assignment index out of range
```

Para que o programa não seja interrompido por erros de execução foram criadas as exceções.

Exceções - introdução

Observe a execução deste programa:

```
mat = [None]*4
for i in range(5):
    mat[i] = i
print(mat)
```

Não há erro de sintaxe, o problema que ocorre é simplesmente que o vetor mat tem 4 posições, 0,1,2,3 e quis-se pôr um valor na posição mat[4] na última iteração do laço.

A resposta obtida é algo como:

```
Programas/prog1.py", line 13, in <module>
    mat[i] = i
IndexError: list assignment index out of range
```

Para que o programa não seja interrompido por erros de execução foram criadas as exceções.

Exceções - introdução

É aconselhável que o desenvolvedor identifique o erro que está sendo trabalhado, caso contrário o python irá prosseguir com o programa desconsiderando qualquer erro cometido.

Exemplo:

```
valor = int(input("Digite um inteiro"))
print("Valor digitado: ", valor)
```

E se o usuário digitar um caracter não numérico o programa vai retornar um erro de valor.

```
valor = int(input("Digite um inteiro: "))#Digite R
ValueError: invalid literal for int() with base 10: 'e'
```

No slide a seguir vamos fazer dois códigos especificando o erro e o deixando em aberto.

Exceções - introdução

É aconselhável que o desenvolvedor identifique o erro que está sendo trabalhado, caso contrário o python irá prosseguir com o programa desconsiderando qualquer erro cometido.

Exemplo:

```
valor = int(input("Digite um inteiro"))
print("Valor digitado: ", valor)
```

E se o usuário digitar um caracter não numérico o programa vai retornar um erro de valor.

```
valor = int(input("Digite um inteiro: "))#Digite R
ValueError: invalid literal for int() with base 10: 'e'
```

No slide a seguir vamos fazer dois códigos especificando o erro e o deixando em aberto.

Exceções - introdução

É aconselhável que o desenvolvedor identifique o erro que está sendo trabalhado, caso contrário o python irá prosseguir com o programa desconsiderando qualquer erro cometido.

Exemplo:

```
valor = int(input("Digite um inteiro"))
print("Valor digitado: ", valor)
```

E se o usuário digitar um caracter não numérico o programa vai retornar um erro de valor.

```
valor = int(input("Digite um inteiro: "))#Digite R
ValueError: invalid literal for int() with base 10: 'e'
```

No slide a seguir vamos fazer dois códigos especificando o erro e o deixando em aberto.

Exceções - introdução

É aconselhável que o desenvolvedor identifique o erro que está sendo trabalhado, caso contrário o python irá prosseguir com o programa desconsiderando qualquer erro cometido.

Exemplo:

```
valor = int(input("Digite um inteiro"))
print("Valor digitado: ", valor)
```

E se o usuário digitar um caracter não numérico o programa vai retornar um erro de valor.

```
valor = int(input("Digite um inteiro: "))#Digite R
ValueError: invalid literal for int() with base 10: 'e'
```

No slide a seguir vamos fazer dois códigos especificando o erro e o deixando em aberto.

Exceções - introdução

Código com erro em aberto:

```
try:
    valor = int(input("Digite um inteiro"))
    print("Valor digitado: ", valor)
except:
    print("É necessário digitar inteiro")
```

Código com erro especificado:

```
try:
    valor = int(input("Digite um inteiro"))
    print("Valor digitado: ", valor)
except ValueError as VErr:
    print("É necessário digitar inteiro")
    print("O erro retornado pelo terminal é: ", VErr)
```

Em ambos os casos o programa não quebra, quando encontra o erro em `try` vai para o bloco de comando de `except` e continua o código que estiver a seguir.
No segundo caso o programa só lança exceção se for um erro de valor.

Exceções - introdução

Código com erro em aberto:

```
try:
    valor = int(input("Digite um inteiro"))
    print("Valor digitado: ", valor)
except:
    print("É necessário digitar inteiro")
```

Código com erro especificado:

```
try:
    valor = int(input("Digite um inteiro"))
    print("Valor digitado: ", valor)
except ValueError as VErr:
    print("É necessário digitar inteiro")
    print("O erro retornado pelo terminal é: ", VErr)
```

Em ambos os casos o programa não quebra, quando encontra o erro em `try` vai para o bloco de comando de `except` e continua o código que estiver a seguir.
No segundo caso o programa só lança exceção se for um erro de valor.

Exceções - introdução

Código com erro em aberto:

```
try:
    valor = int(input("Digite um inteiro"))
    print("Valor digitado: ", valor)
except:
    print("É necessário digitar inteiro")
```

Código com erro especificado:

```
try:
    valor = int(input("Digite um inteiro"))
    print("Valor digitado: ", valor)
except ValueError as VErr:
    print("É necessário digitar inteiro")
    print("O erro retornado pelo terminal é: ", VErr)
```

Em ambos os casos o programa não quebra, quando encontra o erro em `try` vai para o bloco de comando de `except` e continua o código que estiver a seguir.
No segundo caso o programa só lança exceção se for um erro de valor.

Exceções - introdução

Código com erro em aberto:

```
try:
    valor = int(input("Digite um inteiro"))
    print("Valor digitado: ", valor)
except:
    print("É necessário digitar inteiro")
```

Código com erro especificado:

```
try:
    valor = int(input("Digite um inteiro"))
    print("Valor digitado: ", valor)
except ValueError as VErr:
    print("É necessário digitar inteiro")
    print("O erro retornado pelo terminal é: ", VErr)
```

Em ambos os casos o programa não quebra, quando encontra o erro em `try` vai para o bloco de comando de `except` e continua o código que estiver a seguir.
No segundo caso o programa só lança exceção se for um erro de valor.

Exceções - introdução

Código com erro em aberto:

```
try:
    valor = int(input("Digite um inteiro"))
    print("Valor digitado: ", valor)
except:
    print("É necessário digitar inteiro")
```

Código com erro especificado:

```
try:
    valor = int(input("Digite um inteiro"))
    print("Valor digitado: ", valor)
except ValueError as VErr:
    print("É necessário digitar inteiro")
    print("O erro retornado pelo terminal é: ", VErr)
```

Em ambos os casos o programa não quebra, quando encontra o erro em `try` vai para o bloco de comando de `except` e continua o código que estiver a seguir.

No segundo caso o programa só lança exceção se for um erro de valor.

Exceções - introdução

Código com erro em aberto:

```
try:
    valor = int(input("Digite um inteiro"))
    print("Valor digitado: ", valor)
except:
    print("É necessário digitar inteiro")
```

Código com erro especificado:

```
try:
    valor = int(input("Digite um inteiro"))
    print("Valor digitado: ", valor)
except ValueError as VErr:
    print("É necessário digitar inteiro")
    print("O erro retornado pelo terminal é: ", VErr)
```

Em ambos os casos o programa não quebra, quando encontra o erro em `try` vai para o bloco de comando de `except` e continua o código que estiver a seguir.
No segundo caso o programa só lança exceção se for um erro de valor.

Exceções - introdução

Programa com exceção lançada:

```
mat = [None]*4
try:
    for i in range(5):
        mat[i] = i
except IndexError as err:
    print(err, "para i = ",i)
print(mat)
```

As exceções sempre começam com um bloco `try` onde se coloca o bloco de código onde o erro pode acontecer. Caso o erro ocorra é lançada uma exceção com o problema, sem abortar o programa.

No caso apenas o programa imprime o erro que inviabilizaria o programa sem abortá-lo.

A exceção que é lançada é de erro de índice. Rode o programa sem a exceção e com a exceção para comparar as saídas no terminal.

Exceções - introdução

Programa com exceção lançada:

```
mat = [None]*4
try:
    for i in range(5):
        mat[i] = i
except IndexError as err:
    print(err, "para i = ",i)
print(mat)
```

As exceções sempre começam com um bloco `try` onde se coloca o bloco de código onde o erro pode acontecer. Caso o erro ocorra é lançada uma exceção com o problema, sem abortar o programa.

No caso apenas o programa imprime o erro que inviabilizaria o programa sem abortá-lo.

A exceção que é lançada é de erro de índice. Rode o programa sem a exceção e com a exceção para comparar as saídas no terminal.

Exceções - introdução

Programa com exceção lançada:

```
mat = [None]*4
try:
    for i in range(5):
        mat[i] = i
except IndexError as err:
    print(err, "para i = ",i)
print(mat)
```

As exceções sempre começam com um bloco `try` onde se coloca o bloco de código onde o erro pode acontecer. Caso o erro ocorra é lançada uma exceção com o problema, sem abortar o programa.

No caso apenas o programa imprime o erro que inviabilizaria o programa sem abortá-lo.

A exceção que é lançada é de erro de índice. Rode o programa sem a exceção e com a exceção para comparar as saídas no terminal.

Exceções - introdução

Programa com exceção lançada:

```
mat = [None]*4
try:
    for i in range(5):
        mat[i] = i
except IndexError as err:
    print(err, "para i = ",i)
print(mat)
```

As exceções sempre começam com um bloco `try` onde se coloca o bloco de código onde o erro pode acontecer. Caso o erro ocorra é lançada uma exceção com o problema, sem abortar o programa.

No caso apenas o programa imprime o erro que inviabilizaria o programa sem abortá-lo.

A exceção que é lançada é de erro de índice. Rode o programa sem a exceção e com a exceção para comparar as saídas no terminal.

Exceções - introdução

Programa com exceção lançada:

```
mat = [None]*4
try:
    for i in range(5):
        mat[i] = i
except IndexError as err:
    print(err, "para i = ",i)
print(mat)
```

As exceções sempre começam com um bloco `try` onde se coloca o bloco de código onde o erro pode acontecer. Caso o erro ocorra é lançada uma exceção com o problema, sem abortar o programa.

No caso apenas o programa imprime o erro que inviabilizaria o programa sem abortá-lo.

A exceção que é lançada é de erro de índice. Rode o programa sem a exceção e com a exceção para comparar as saídas no terminal.

Sumário

Exceções - Exemplo

Um exemplo útil de uso seria fazer um programa que recebesse do usuário uma quantidade desconhecida de números inteiros, imprimisse no final a quantidade de inteiros, a soma deles e a média. O fim do programa deve ocorrer se o usuário digitar enter sem nenhum número.

Faça este programa com os modos:

1. Sem nenhuma exceção. Quando executar insira os dados sem nenhuma entrada errada (todos os números inteiros) e no final digite enter somente para sair. ver Prog3.py.
   Digite algo errado, como ao invés de um número um caracter alfabético.
2. Lançando uma exceção para leitura de número que não pode ser convertido em inteiro, mas sem mostrar o erro. Use pass no bloco da exceção.ver Prog4.py
   Observe que caso o valor digitado não seja um inteiro a conversão não é realizada e o valor do termo lido permanece inalterado.
3. Lançando uma exceção para leitura de número que não pode ser convertido em inteiro, mostrando o erro no terminal, e esperando do usuário que escreva corretamente um inteiro. ver Prog5.py

Exceções - Exemplo

Um exemplo útil de uso seria fazer um programa que recebesse do usuário uma quantidade desconhecida de números inteiros, imprimisse no final a quantidade de inteiros, a soma deles e a média. O fim do programa deve ocorrer se o usuário digitar enter sem nenhum número.
Faça este programa com os modos:

1. 1.) Sem nenhuma exceção. Quando executar insira os dados sem nenhuma entrada errada (todos os números inteiros) e no final digite enter somente para sair. ver Prog3.py.
   Digite algo errado, como ao invés de um número um caracter alfabético.
2. Lançando uma exceção para leitura de número que não pode ser convertido em inteiro, mas sem mostrar o erro. Use pass no bloco da exceção.ver Prog4.py
   Observe que caso o valor digitado não seja um inteiro a conversão não é realizada e o valor do termo lido permanece inalterado.
3. Lançando uma exceção para leitura de número que não pode ser convertido em inteiro, mostrando o erro no terminal, e esperando do usuário que escreva corretamente um inteiro. ver Prog5.py

Exceções - Exemplo

Um exemplo útil de uso seria fazer um programa que recebesse do usuário uma quantidade desconhecida de números inteiros, imprimisse no final a quantidade de inteiros, a soma deles e a média. O fim do programa deve ocorrer se o usuário digitar enter sem nenhum número.
Faça este programa com os modos:

1. 1.) Sem nenhuma exceção. Quando executar insira os dados sem nenhuma entrada errada (todos os números inteiros) e no final digite enter somente para sair. ver Prog3.py.
   Digite algo errado, como ao invés de um número um caracter alfabético.
2. Lançando uma exceção para leitura de número que não pode ser convertido em inteiro, mas sem mostrar o erro. Use pass no bloco da exceção.ver Prog4.py
   Observe que caso o valor digitado não seja um inteiro a conversão não é realizada e o valor do termo lido permanece inalterado.
3. Lançando uma exceção para leitura de número que não pode ser convertido em inteiro, mostrando o erro no terminal, e esperando do usuário que escreva corretamente um inteiro. ver Prog5.py

Exceções - Exemplo

Um exemplo útil de uso seria fazer um programa que recebesse do usuário uma quantidade desconhecida de números inteiros, imprimisse no final a quantidade de inteiros, a soma deles e a média. O fim do programa deve ocorrer se o usuário digitar enter sem nenhum número.
Faça este programa com os modos:

1. 1.) Sem nenhuma exceção. Quando executar insira os dados sem nenhuma entrada errada (todos os números inteiros) e no final digite enter somente para sair. ver Prog3.py.
   Digite algo errado, como ao invés de um número um caracter alfabético.
2. Lançando uma exceção para leitura de número que não pode ser convertido em inteiro, mas sem mostrar o erro. Use pass no bloco da exceção.ver Prog4.py
   Observe que caso o valor digitado não seja um inteiro a conversão não é realizada e o valor do termo lido permanece inalterado.
3. Lançando uma exceção para leitura de número que não pode ser convertido em inteiro, mostrando o erro no terminal, e esperando do usuário que escreva corretamente um inteiro. ver Prog5.py

Exceções - Exemplo

Um exemplo útil de uso seria fazer um programa que recebesse do usuário uma quantidade desconhecida de números inteiros, imprimisse no final a quantidade de inteiros, a soma deles e a média. O fim do programa deve ocorrer se o usuário digitar enter sem nenhum número.
Faça este programa com os modos:

1. Sem nenhuma exceção. Quando executar insira os dados sem nenhuma entrada errada (todos os números inteiros) e no final digite enter somente para sair. ver Prog3.py.
   Digite algo errado, como ao invés de um número um caracter alfabético.
2. Lançando uma exceção para leitura de número que não pode ser convertido em inteiro, mas sem mostrar o erro. Use pass no bloco da exceção.ver Prog4.py
   Observe que caso o valor digitado não seja um inteiro a conversão não é realizada e o valor do termo lido permanece inalterado.
3. Lançando uma exceção para leitura de número que não pode ser convertido em inteiro, mostrando o erro no terminal, e esperando do usuário que escreva corretamente um inteiro. ver Prog5.py

Exceções - Exemplo

Um exemplo útil de uso seria fazer um programa que recebesse do usuário uma quantidade desconhecida de números inteiros, imprimisse no final a quantidade de inteiros, a soma deles e a média. O fim do programa deve ocorrer se o usuário digitar enter sem nenhum número.
Faça este programa com os modos:

1. Sem nenhuma exceção. Quando executar insira os dados sem nenhuma entrada errada (todos os números inteiros) e no final digite enter somente para sair. ver Prog3.py.
   Digite algo errado, como ao invés de um número um caracter alfabético.
2. Lançando uma exceção para leitura de número que não pode ser convertido em inteiro, mas sem mostrar o erro. Use pass no bloco da exceção.ver Prog4.py
   Observe que caso o valor digitado não seja um inteiro a conversão não é realizada e o valor do termo lido permanece inalterado.
3. Lançando uma exceção para leitura de número que não pode ser convertido em inteiro, mostrando o erro no terminal, e esperando do usuário que escreva corretamente um inteiro. ver Prog5.py

Exceções - Exemplo

Um exemplo útil de uso seria fazer um programa que recebesse do usuário uma quantidade desconhecida de números inteiros, imprimisse no final a quantidade de inteiros, a soma deles e a média. O fim do programa deve ocorrer se o usuário digitar enter sem nenhum número.
Faça este programa com os modos:

1. Sem nenhuma exceção. Quando executar insira os dados sem nenhuma entrada errada (todos os números inteiros) e no final digite enter somente para sair. ver Prog3.py.
   Digite algo errado, como ao invés de um número um caracter alfabético.
2. Lançando uma exceção para leitura de número que não pode ser convertido em inteiro, mas sem mostrar o erro. Use pass no bloco da exceção.ver Prog4.py
   Observe que caso o valor digitado não seja um inteiro a conversão não é realizada e o valor do termo lido permanece inalterado.
3. Lançando uma exceção para leitura de número que não pode ser convertido em inteiro, mostrando o erro no terminal, e esperando do usuário que escreva corretamente um inteiro. ver Prog5.py

Sumário

Exceções nativas do Python

Experimente no script Prog5.py digitar enter logo no primeiro número e veja o que ocorre.

O programa não funciona porque há uma operação matemática proibida, divisão por zero. Temos duas formas de proceder:

1. inserir um comando if verificando se algum item foi digitado.
2. criar uma exceção.

Há algumas exceções nativas do python, são chamadas exceções nativas (ou exceções built-in). Uma delas é para previnir divisão por zero. Inserindo no código, após uma chamada a `try:`

```
try:
    bloco de codigo
except ZeroDivisionError:
    pass
```

O programa é executado, deixando de fazer a operação indicada.

Se quisermos saber o erro que apareceria no terminal podemos iniciar uma variável na cláusula `except:`

```
except ZeroDivisionError as erro1:
```

e no bloco de código da exceção pedir a impressão de `erro1`. ver Prog6.py

Exceções nativas do Python

Experimente no script Prog5.py digitar enter logo no primeiro número e veja o que ocorre.

O programa não funciona porque há uma operação matemática proibida, divisão por zero. Temos duas formas de proceder:

1. inserir um comando if verificando se algum item foi digitado.
2. criar uma exceção.

Há algumas exceções nativas do python, são chamadas exceções nativas (ou exceções built-in). Uma delas é para previnir divisão por zero. Inserindo no código, após uma chamada a `try:`

```
try:
    bloco de codigo
except ZeroDivisionError:
    pass
```

O programa é executado, deixando de fazer a operação indicada.

Se quisermos saber o erro que apareceria no terminal podemos iniciar uma variável na cláusula `except:`

```
except ZeroDivisionError as erro1:
```

e no bloco de código da exceção pedir a impressão de `erro1`. ver Prog6.py

Exceções nativas do Python

Experimente no script Prog5.py digitar enter logo no primeiro número e veja o que ocorre.

O programa não funciona porque há uma operação matemática proibida, divisão por zero. Temos duas formas de proceder:

1. inserir um comando if verificando se algum item foi digitado.
2. criar uma exceção.

Há algumas exceções nativas do python, são chamadas exceções nativas (ou exceções built-in). Uma delas é para previnir divisão por zero. Inserindo no código, após uma chamada a `try:`

```
try:
    bloco de codigo
except ZeroDivisionError:
    pass
```

O programa é executado, deixando de fazer a operação indicada.

Se quisermos saber o erro que apareceria no terminal podemos iniciar uma variável na cláusula `except:`

```
except ZeroDivisionError as erro1:
```

e no bloco de código da exceção pedir a impressão de `erro1`. ver Prog6.py

Exceções nativas do Python

Experimente no script Prog5.py digitar enter logo no primeiro número e veja o que ocorre.

O programa não funciona porque há uma operação matemática proibida, divisão por zero. Temos duas formas de proceder:

1. inserir um comando if verificando se algum item foi digitado.
2. criar uma exceção.

Há algumas exceções nativas do python, são chamadas exceções nativas (ou exceções built-in). Uma delas é para previnir divisão por zero. Inserindo no código, após uma chamada a `try:`

```
try:
    bloco de codigo
except ZeroDivisionError:
    pass
```

O programa é executado, deixando de fazer a operação indicada.

Se quisermos saber o erro que apareceria no terminal podemos iniciar uma variável na cláusula `except:`

```
except ZeroDivisionError as erro1:
```

e no bloco de código da exceção pedir a impressão de `erro1`. ver Prog6.py

Exceções nativas do Python

Experimente no script Prog5.py digitar enter logo no primeiro número e veja o que ocorre.

O programa não funciona porque há uma operação matemática proibida, divisão por zero. Temos duas formas de proceder:

1. inserir um comando if verificando se algum item foi digitado.
2. criar uma exceção.

Há algumas exceções nativas do python, são chamadas exceções nativas (ou exceções built-in). Uma delas é para previnir divisão por zero. Inserindo no código, após uma chamada a `try:`

```
try:
    bloco de codigo
except ZeroDivisionError:
    pass
```

O programa é executado, deixando de fazer a operação indicada.

Se quisermos saber o erro que apareceria no terminal podemos iniciar uma variável na cláusula `except:`

```
except ZeroDivisionError as erro1:
```

e no bloco de código da exceção pedir a impressão de `erro1`. ver Prog6.py

Exceções nativas do Python

Experimente no script Prog5.py digitar enter logo no primeiro número e veja o que ocorre.

O programa não funciona porque há uma operação matemática proibida, divisão por zero. Temos duas formas de proceder:

1. inserir um comando if verificando se algum item foi digitado.
2. criar uma exceção.

Há algumas exceções nativas do python, são chamadas exceções nativas (ou exceções built-in). Uma delas é para previnir divisão por zero. Inserindo no código, após uma chamada a `try:`

```
try:
    bloco de codigo
except ZeroDivisionError:
    pass
```

O programa é executado, deixando de fazer a operação indicada.

Se quisermos saber o erro que apareceria no terminal podemos iniciar uma variável na cláusula `except:`

```
except ZeroDivisionError as erro1:
```

e no bloco de código da exceção pedir a impressão de `erro1`. ver Prog6.py

Exceções nativas do Python

Experimente no script Prog5.py digitar enter logo no primeiro número e veja o que ocorre.

O programa não funciona porque há uma operação matemática proibida, divisão por zero. Temos duas formas de proceder:

1. inserir um comando if verificando se algum item foi digitado.
2. criar uma exceção.

Há algumas exceções nativas do python, são chamadas exceções nativas (ou exceções built-in). Uma delas é para previnir divisão por zero. Inserindo no código, após uma chamada a `try:`

```
try:
    bloco de codigo
except ZeroDivisionError:
   pass
```

O programa é executado, deixando de fazer a operação indicada.

Se quisermos saber o erro que apareceria no terminal podemos iniciar uma variável na cláusula `except:`

```
except ZeroDivisionError as erro1:
```

e no bloco de código da exceção pedir a impressão de `erro1`. ver Prog6.py

Exceções nativas do Python

Experimente no script Prog5.py digitar enter logo no primeiro número e veja o que ocorre.

O programa não funciona porque há uma operação matemática proibida, divisão por zero. Temos duas formas de proceder:

1. inserir um comando if verificando se algum item foi digitado.
2. criar uma exceção.

Há algumas exceções nativas do python, são chamadas exceções nativas (ou exceções built-in). Uma delas é para previnir divisão por zero. Inserindo no código, após uma chamada a `try:`

```
try:
    bloco de codigo
except ZeroDivisionError:
   pass
```

O programa é executado, deixando de fazer a operação indicada.

Se quisermos saber o erro que apareceria no terminal podemos iniciar uma variável na cláusula `except:`

```
except ZeroDivisionError as erro1:
```

e no bloco de código da exceção pedir a impressão de `erro1`. ver Prog6.py

Exceções nativas do Python

Outra exceção nativa muito importante é a verificação de fim do arquivo (*End Of File - EOF*) em inglês.

Um programa interessante seria refazer Prog6.py considerando leitura por redirecionamento em arquivo com valores inteiros dispostos em várias linhas.

Este é um exemplo de uso obrigatório da cláusula `try-except`, e vamos ver alguns exemplos na plataforma *Beecrowd* que ilustram esta forma de implementação.

Exceções nativas do Python

Outra exceção nativa muito importante é a verificação de fim do arquivo (*End Of File - EOF*) em inglês.

Um programa interessante seria refazer Prog6.py considerando leitura por redirecionamento em arquivo com valores inteiros dispostos em várias linhas.

Este é um exemplo de uso obrigatório da cláusula `try-except`, e vamos ver alguns exemplos na plataforma *Beecrowd* que ilustram esta forma de implementação.

Exceções nativas do Python

Outra exceção nativa muito importante é a verificação de fim do arquivo (*End Of File - EOF*) em inglês.

Um programa interessante seria refazer Prog6.py considerando leitura por redirecionamento em arquivo com valores inteiros dispostos em várias linhas.

Este é um exemplo de uso obrigatório da cláusula `try-except`, e vamos ver alguns exemplos na plataforma *Beecrowd* que ilustram esta forma de implementação.

Sumário

Try - except - else do Python

Usamos a cláusula `else` geralmente quando queremos verificar uma garantia de não passagem pela exceção, mas pode ser usada para qualquer propósito.

Vamos seguir o roteiro abaixo:

- Veja Prog7.py para incluir a clausula else apenas para dar informação de validação para verificação de divisão por zero.
- Veja o Prog8.py que insere uma variável que é definida no interior de um bloco try que pode não ser executado.
- Veja o Prog9.py que corrige o problema anterior utilizando exceptś alinhados, com o viés de que somente o último except pode ter cláusula else.
- Por último Prog10.py separa os blocos try-except para que cada um possa ter uma cláusula else para si.

Try - except - else do Python

Usamos a cláusula `else` geralmente quando queremos verificar uma garantia de não passagem pela exceção, mas pode ser usada para qualquer propósito.

Vamos seguir o roteiro abaixo:

- Veja Prog7.py para incluir a clausula else apenas para dar informação de validação para verificação de divisão por zero.
- Veja o Prog8.py que insere uma variável que é definida no interior de um bloco try que pode não ser executado.
- Veja o Prog9.py que corrige o problema anterior utilizando exceptś alinhados, com o viés de que somente o último except pode ter cláusula else.
- Por último Prog10.py separa os blocos try-except para que cada um possa ter uma cláusula else para si.

Try - except - else do Python

Usamos a cláusula `else` geralmente quando queremos verificar uma garantia de não passagem pela exceção, mas pode ser usada para qualquer propósito.

Vamos seguir o roteiro abaixo:

- Veja Prog7.py para incluir a clausula else apenas para dar informação de validação para verificação de divisão por zero.
- Veja o Prog8.py que insere uma variável que é definida no interior de um bloco try que pode não ser executado.
- Veja o Prog9.py que corrige o problema anterior utilizando exceptś alinhados, com o viés de que somente o último except pode ter cláusula else.
- Por último Prog10.py separa os blocos try-except para que cada um possa ter uma cláusula else para si.

Try - except - else do Python

Usamos a cláusula `else` geralmente quando queremos verificar uma garantia de não passagem pela exceção, mas pode ser usada para qualquer propósito.

Vamos seguir o roteiro abaixo:

- Veja Prog7.py para incluir a clausula else apenas para dar informação de validação para verificação de divisão por zero.
- Veja o Prog8.py que insere uma variável que é definida no interior de um bloco try que pode não ser executado.
- Veja o Prog9.py que corrige o problema anterior utilizando exceptś alinhados, com o viés de que somente o último except pode ter cláusula else.
- Por último Prog10.py separa os blocos try-except para que cada um possa ter uma cláusula else para si.

Try - except - else do Python

Usamos a cláusula `else` geralmente quando queremos verificar uma garantia de não passagem pela exceção, mas pode ser usada para qualquer propósito.

Vamos seguir o roteiro abaixo:

- Veja Prog7.py para incluir a clausula else apenas para dar informação de validação para verificação de divisão por zero.
- Veja o Prog8.py que insere uma variável que é definida no interior de um bloco try que pode não ser executado.
- Veja o Prog9.py que corrige o problema anterior utilizando exceptś alinhados, com o viés de que somente o último except pode ter cláusula else.
- Por último Prog10.py separa os blocos try-except para que cada um possa ter uma cláusula else para si.

Sumário

Cláusula finally do Python

A cláusula finally é usada pouco por programadores, mas faz parte das exceções.

A cláusula serve continuar a execução do código, independentemente do que ocorre no bloco try.

O script Prog11.py ilustra a execução do bloco finally.

Perceba que:

Quando a exceção é encontrada, os comandos depois da exceção não são executados.

A linha:

```
print("Interior do try depois da exceção", x + y)
```

não é executada se houver uma exceção antes dela.

Coloque uma exceção no código anterior e verifique que a linha depois do bloco try é executada.

Cláusula finally do Python

A cláusula finally é usada pouco por programadores, mas faz parte das exceções.
A cláusula serve continuar a execução do código, independentemente do que ocorre no bloco try.
O script Prog11.py ilustra a execução do bloco finally.

Perceba que:
Quando a exceção é encontrada, os comandos depois da exceção não são executados.
A linha:

```
print("Interior do try depois da exceção", x + y)
```

não é executada se houver uma exceção antes dela.
Coloque uma exceção no código anterior e verifique que a linha depois do bloco try é executada.

Cláusula finally do Python

A cláusula finally é usada pouco por programadores, mas faz parte das exceções.
A cláusula serve continuar a execução do código, independentemente do que ocorre no bloco try.
O script Prog11.py ilustra a execução do bloco finally.

Perceba que:
Quando a exceção é encontrada, os comandos depois da exceção não são executados.
A linha:

```
print("Interior do try depois da exceção", x + y)
```

não é executada se houver uma exceção antes dela.
Coloque uma exceção no código anterior e verifique que a linha depois do bloco try é executada.

Cláusula finally do Python

A cláusula finally é usada pouco por programadores, mas faz parte das exceções.
A cláusula serve continuar a execução do código, independentemente do que ocorre no bloco try.
O script Prog11.py ilustra a execução do bloco finally.

Perceba que:
Quando a exceção é encontrada, os comandos depois da exceção não são executados.
A linha:

```
print("Interior do try depois da exceção", x + y)
```

não é executada se houver uma exceção antes dela.
Coloque uma exceção no código anterior e verifique que a linha depois do bloco try é executada.

Cláusula finally do Python

A cláusula finally é usada pouco por programadores, mas faz parte das exceções.
A cláusula serve continuar a execução do código, independentemente do que ocorre no bloco try.
O script Prog11.py ilustra a execução do bloco finally.

Perceba que:
Quando a exceção é encontrada, os comandos depois da exceção não são executados.
A linha:

```
print("Interior do try depois da exceção", x + y)
```

não é executada se houver uma exceção antes dela.
Coloque uma exceção no código anterior e verifique que a linha depois do bloco try é executada.

Cláusula finally do Python

A cláusula finally é usada pouco por programadores, mas faz parte das exceções.
A cláusula serve continuar a execução do código, independentemente do que ocorre no bloco try.
O script Prog11.py ilustra a execução do bloco finally.

Perceba que:
Quando a exceção é encontrada, os comandos depois da exceção não são executados.
A linha:

```
print("Interior do try depois da exceção", x + y)
```

não é executada se houver uma exceção antes dela.
Coloque uma exceção no código anterior e verifique que a linha depois do bloco try é executada.