Notas de aula para OBI
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

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

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.

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.

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.

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.

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.

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.

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.

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.

Veja Prog7.py para verificar a clausula else apenas para dar informação de validação e o script Prog8.py para usar uma variável que pode ou não ser criada pela utilização de um bloco try-except.

Verifique o script Prog9.py usando uma terceira exceção para corrigir o problema do script Prog8.py.

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.

Veja Prog7.py para verificar a clausula else apenas para dar informação de validação e o script Prog8.py para usar uma variável que pode ou não ser criada pela utilização de um bloco try-except.

Verifique o script Prog9.py usando uma terceira exceção para corrigir o problema do script Prog8.py.

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.

Veja Prog7.py para verificar a clausula else apenas para dar informação de validação e o script Prog8.py para usar uma variável que pode ou não ser criada pela utilização de um bloco try-except.

Verifique o script Prog9.py usando uma terceira exceção para corrigir o problema do script Prog8.py.

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.

```
x = 2
y = 0
try:
    Div = x / y
finally:
    print("x+y = ", x+y)
```

Nada além do bloco finally é feito, e o programa é abortado.

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.

```
x = 2
y = 0
try:
    Div = x / y
finally:
    print("x+y = ", x+y)
```

Nada além do bloco finally é feito, e o programa é abortado.

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.

```
x = 2
y = 0
try:
    Div = x / y
finally:
    print("x+y = ", x+y)
```

Nada além do bloco finally é feito, e o programa é abortado.

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.

```
x = 2
y = 0
try:
    Div = x / y
finally:
    print("x+y = ", x+y)
```

Nada além do bloco finally é feito, e o programa é abortado.

Sumário

Exceções built-ins

Exceções nativas, ou exceções *built-in* são exceções que já são definidas pela linguagem e que podem ser usadas pelo programador para evitar quebras do programa.

Há exceções de vários tipos, para verificar tecla digitada errado, para índice errado em vetores, para controle de arquivo etc.

Para obter todas as exceções nativas do python basta ir em:
`https://docs.python.org/3/library/exceptions.html`

Uma das exceções mais usadas é EOFError, que se usa para verificar se o fim de um arquivo foi alcançado.

Como exemplo, faça um arquivo com várias linhas de entrada, cada uma com dois inteiros. Leia o arquivo usando redirecionamento e por fim imprima em arquivo (também usando redirecionamento) o total de linhas lidas, a soma e a média da divisão do primeiro pelo segundo valor de cada linha, quando não for uma divisão por zero ou não for uma string não numérica.

É importante saber que uma cláusula try pode ter várias cláusulas except.

Exceções built-ins

Exceções nativas, ou exceções *built-in* são exceções que já são definidas pela linguagem e que podem ser usadas pelo programador para evitar quebras do programa.

Há exceções de vários tipos, para verificar tecla digitada errado, para índice errado em vetores, para controle de arquivo etc.

Para obter todas as exceções nativas do python basta ir em:
`https://docs.python.org/3/library/exceptions.html`

Uma das exceções mais usadas é EOFError, que se usa para verificar se o fim de um arquivo foi alcançado.

Como exemplo, faça um arquivo com várias linhas de entrada, cada uma com dois inteiros. Leia o arquivo usando redirecionamento e por fim imprima em arquivo (também usando redirecionamento) o total de linhas lidas, a soma e a média da divisão do primeiro pelo segundo valor de cada linha, quando não for uma divisão por zero ou não for uma string não numérica.

É importante saber que uma cláusula try pode ter várias cláusulas except.

Exceções built-ins

Exceções nativas, ou exceções *built-in* são exceções que já são definidas pela linguagem e que podem ser usadas pelo programador para evitar quebras do programa.

Há exceções de vários tipos, para verificar tecla digitada errado, para índice errado em vetores, para controle de arquivo etc.

Para obter todas as exceções nativas do python basta ir em:
`https://docs.python.org/3/library/exceptions.html`

Uma das exceções mais usadas é EOFError, que se usa para verificar se o fim de um arquivo foi alcançado.

Como exemplo, faça um arquivo com várias linhas de entrada, cada uma com dois inteiros. Leia o arquivo usando redirecionamento e por fim imprima em arquivo (também usando redirecionamento) o total de linhas lidas, a soma e a média da divisão do primeiro pelo segundo valor de cada linha, quando não for uma divisão por zero ou não for uma string não numérica.

É importante saber que uma cláusula try pode ter várias cláusulas except.

Exceções built-ins

Exceções nativas, ou exceções *built-in* são exceções que já são definidas pela linguagem e que podem ser usadas pelo programador para evitar quebras do programa.

Há exceções de vários tipos, para verificar tecla digitada errado, para índice errado em vetores, para controle de arquivo etc.

Para obter todas as exceções nativas do python basta ir em:
`https://docs.python.org/3/library/exceptions.html`

Uma das exceções mais usadas é EOFError, que se usa para verificar se o fim de um arquivo foi alcançado.

Como exemplo, faça um arquivo com várias linhas de entrada, cada uma com dois inteiros. Leia o arquivo usando redirecionamento e por fim imprima em arquivo (também usando redirecionamento) o total de linhas lidas, a soma e a média da divisão do primeiro pelo segundo valor de cada linha, quando não for uma divisão por zero ou não for uma string não numérica.

É importante saber que uma cláusula try pode ter várias cláusulas except.

Exceções built-ins

Exceções nativas, ou exceções *built-in* são exceções que já são definidas pela linguagem e que podem ser usadas pelo programador para evitar quebras do programa.

Há exceções de vários tipos, para verificar tecla digitada errado, para índice errado em vetores, para controle de arquivo etc.

Para obter todas as exceções nativas do python basta ir em:
`https://docs.python.org/3/library/exceptions.html`

Uma das exceções mais usadas é EOFError, que se usa para verificar se o fim de um arquivo foi alcançado.

Como exemplo, faça um arquivo com várias linhas de entrada, cada uma com dois inteiros. Leia o arquivo usando redirecionamento e por fim imprima em arquivo (também usando redirecionamento) o total de linhas lidas, a soma e a média da divisão do primeiro pelo segundo valor de cada linha, quando não for uma divisão por zero ou não for uma string não numérica.

É importante saber que uma cláusula try pode ter várias cláusulas except.

Exceções built-ins

Exceções nativas, ou exceções *built-in* são exceções que já são definidas pela linguagem e que podem ser usadas pelo programador para evitar quebras do programa.

Há exceções de vários tipos, para verificar tecla digitada errado, para índice errado em vetores, para controle de arquivo etc.

Para obter todas as exceções nativas do python basta ir em:
`https://docs.python.org/3/library/exceptions.html`

Uma das exceções mais usadas é EOFError, que se usa para verificar se o fim de um arquivo foi alcançado.

Como exemplo, faça um arquivo com várias linhas de entrada, cada uma com dois inteiros. Leia o arquivo usando redirecionamento e por fim imprima em arquivo (também usando redirecionamento) o total de linhas lidas, a soma e a média da divisão do primeiro pelo segundo valor de cada linha, quando não for uma divisão por zero ou não for uma string não numérica.

É importante saber que uma cláusula try pode ter várias cláusulas except.

Exceções built-ins

A exceção EOFError verifica se a leitura de um arquivo chegou ao final. Se tiver chegado, lança esta exceção. Faça um programa (Prog11.py) que simplesmente leia as linhas de um arquivo e pare quando chegar ao final do arquivo.
Explique o motivo de, no script Prog11.py (use redirecionamento abrindo o arquivo `data_in.dat`):

1. O número de linhas lidas foi maior do que o do arquivo no arquivo `data_in.dat`.
2. Trocar de lugar as duas except usadas não faz diferença no programa.

Faça um script completo para resolver o problema usando uma exceção que ignora divisão por zero e de leitura não numérica de uma linha. Veja o arquivo Prog12.py.

Faça o problema 2540 do beecrowd.

Exceções built-ins

A exceção EOFError verifica se a leitura de um arquivo chegou ao final. Se tiver chegado, lança esta exceção. Faça um programa (Prog11.py) que simplesmente leia as linhas de um arquivo e pare quando chegar ao final do arquivo.
Explique o motivo de, no script Prog11.py (use redirecionamento abrindo o arquivo `data_in.dat`):

1. O número de linhas lidas foi maior do que o do arquivo no arquivo `data_in.dat`.
2. Trocar de lugar as duas except usadas não faz diferença no programa.

Faça um script completo para resolver o problema usando uma exceção que ignora divisão por zero e de leitura não numérica de uma linha. Veja o arquivo Prog12.py.

Faça o problema 2540 do beecrowd.

Exceções built-ins

A exceção EOFError verifica se a leitura de um arquivo chegou ao final. Se tiver chegado, lança esta exceção. Faça um programa (Prog11.py) que simplesmente leia as linhas de um arquivo e pare quando chegar ao final do arquivo.
Explique o motivo de, no script Prog11.py (use redirecionamento abrindo o arquivo `data_in.dat`):

1. O número de linhas lidas foi maior do que o do arquivo no arquivo `data_in.dat`.
2. Trocar de lugar as duas except usadas não faz diferença no programa.

Faça um script completo para resolver o problema usando uma exceção que ignora divisão por zero e de leitura não numérica de uma linha. Veja o arquivo Prog12.py.

Faça o problema 2540 do beecrowd.

Exceções built-ins

A exceção EOFError verifica se a leitura de um arquivo chegou ao final. Se tiver chegado, lança esta exceção. Faça um programa (Prog11.py) que simplesmente leia as linhas de um arquivo e pare quando chegar ao final do arquivo.
Explique o motivo de, no script Prog11.py (use redirecionamento abrindo o arquivo `data_in.dat`):

1. O número de linhas lidas foi maior do que o do arquivo no arquivo `data_in.dat`.
2. Trocar de lugar as duas except usadas não faz diferença no programa.

Faça um script completo para resolver o problema usando uma exceção que ignora divisão por zero e de leitura não numérica de uma linha. Veja o arquivo Prog12.py.

Faça o problema 2540 do beecrowd.

Exceções built-ins

A exceção EOFError verifica se a leitura de um arquivo chegou ao final. Se tiver chegado, lança esta exceção. Faça um programa (Prog11.py) que simplesmente leia as linhas de um arquivo e pare quando chegar ao final do arquivo.
Explique o motivo de, no script Prog11.py (use redirecionamento abrindo o arquivo `data_in.dat`):

1. O número de linhas lidas foi maior do que o do arquivo no arquivo `data_in.dat`.
2. Trocar de lugar as duas except usadas não faz diferença no programa.

Faça um script completo para resolver o problema usando uma exceção que ignora divisão por zero e de leitura não numérica de uma linha. Veja o arquivo Prog12.py.

Faça o problema 2540 do beecrowd.

Exceções built-ins

A exceção EOFError verifica se a leitura de um arquivo chegou ao final. Se tiver chegado, lança esta exceção. Faça um programa (Prog11.py) que simplesmente leia as linhas de um arquivo e pare quando chegar ao final do arquivo.
Explique o motivo de, no script Prog11.py (use redirecionamento abrindo o arquivo `data_in.dat`):

1. O número de linhas lidas foi maior do que o do arquivo no arquivo `data_in.dat`.
2. Trocar de lugar as duas except usadas não faz diferença no programa.

Faça um script completo para resolver o problema usando uma exceção que ignora divisão por zero e de leitura não numérica de uma linha. Veja o arquivo Prog12.py.

Faça o problema 2540 do beecrowd.

Exceção TypeError

TypeError é uma exceção lançada quando uma operação incorreta é realizada, como usar o operador + para somar uma string e um inteiro.

Exemplo:

```
num = '4'
num2 = 2
print(num+num2)
```

Uma versão corrigida é dada por:

```
num = '4'
num2 = 2
try:
    print(num+num2)
except TypeError as Erro:
    print(Erro)
    print("Dados somados incompatíveis")
print("Final")
```

Um outro exemplo seria chamar uma função que não existe:

```
nome = "Carlos"
print(Carlos())
```

Exceção TypeError

TypeError é uma exceção lançada quando uma operação incorreta é realizada, como usar o operador + para somar uma string e um inteiro.

Exemplo:

```
num = '4'
num2 = 2
print(num+num2)
```

Uma versão corrigida é dada por:

```
num = '4'
num2 = 2
try:
    print(num+num2)
except TypeError as Erro:
    print(Erro)
    print("Dados somados incompatíveis")
print("Final")
```

Um outro exemplo seria chamar uma função que não existe:

```
nome = "Carlos"
print(Carlos())
```

Exceção TypeError

TypeError é uma exceção lançada quando uma operação incorreta é realizada, como usar o operador + para somar uma string e um inteiro.

Exemplo:

```
num = '4'
num2 = 2
print(num+num2)
```

Uma versão corrigida é dada por:

```
num = '4'
num2 = 2
try:
    print(num+num2)
except TypeError as Erro:
    print(Erro)
    print("Dados somados incompatíveis")
print("Final")
```

Um outro exemplo seria chamar uma função que não existe:

```
nome = "Carlos"
print(Carlos())
```

Exceção TypeError

TypeError é uma exceção lançada quando uma operação incorreta é realizada, como usar o operador + para somar uma string e um inteiro.

```
Exemplo:
num = '4'
num2 = 2
print(num+num2)
```

Uma versão corrigida é dada por:

```
num = '4'
num2 = 2
try:
    print(num+num2)
except TypeError as Erro:
    print(Erro)
    print("Dados somados incompatíveis")
print("Final")
```

Um outro exemplo seria chamar uma função que não existe:

```
nome = "Carlos"
print(Carlos())
```

Sumário

Exceções criadas por usuário

Um programador pode ele próprio fazer uma espécie de gerenciamento de erros, por exemplo, de usuários alimentando dados no programa.

Desta forma o bloco de código num comando `try` é executado, e se este não gerar um erro o programa prossegue, do contrário adentra o bloco seguido pelo comando `except`:

```
var1 = '1'
try:
    var1 = var1 + 1
except:
    print(var1, " não é um número, refaça operação.")
```

Uma boa saída seria:

```
try:
    var2 = var1 + 1
except:
    var2 = int(var1) + 1
print(var2)
```

Exceções criadas por usuário

Um programador pode ele próprio fazer uma espécie de gerenciamento de erros, por exemplo, de usuários alimentando dados no programa.
Desta forma o bloco de código num comando `try` é executado, e se este não gerar um erro o programa prossegue, do contrário adentra o bloco seguido pelo comando `except`:

```
var1 = '1'
try:
    var1 = var1 + 1
except:
    print(var1, " não é um número, refaça operação.")
```

Uma boa saída seria:

```
try:
    var2 = var1 + 1
except:
    var2 = int(var1) + 1
print(var2)
```

Exceções criadas por usuário

Um programador pode ele próprio fazer uma espécie de gerenciamento de erros, por exemplo, de usuários alimentando dados no programa.
Desta forma o bloco de código num comando `try` é executado, e se este não gerar um erro o programa prossegue, do contrário adentra o bloco seguido pelo comando `except`:

```
var1 = '1'
try:
    var1 = var1 + 1
except:
    print(var1, " não é um número, refaça operação.")
```

Uma boa saída seria:

```
try:
    var2 = var1 + 1
except:
    var2 = int(var1) + 1
print(var2)
```

Exceções criadas por usuário

Um programador pode ele próprio fazer uma espécie de gerenciamento de erros, por exemplo, de usuários alimentando dados no programa.
Desta forma o bloco de código num comando `try` é executado, e se este não gerar um erro o programa prossegue, do contrário adentra o bloco seguido pelo comando `except`:

```
var1 = '1'
try:
    var1 = var1 + 1
except:
    print(var1, " não é um número, refaça operação.")
```

Uma boa saída seria:

```
try:
    var2 = var1 + 1
except:
    var2 = int(var1) + 1
print(var2)
```

Sumário

BigDigits

Faça um script que leia do terminal, já na chamada do programa, um número escrito pelo usuário e imprima este número usando uma lista de strings para cada algarismo com * e espaços.

Exemplo de algarismo:

```
Dois = [" *** ",
        "*   *",
        "    * ",
        "   *  ",
        "  *   ",
        "*     ",
        "*****"]
```

Você deve considerar duas exceções:

1. O usuário não digitou nenhum número na chamada.
2. O usuário digitou um caracter não numérico na chamada.

Utilize a biblioteca sys (inclua `import sys` logo abaixo da linha shebang) para obter o número digitado usando sys.argv[1].

Exemplo de entrada e saida:

```
louisaugusto@louisaugusto-Desktop:~$ python3 bigdigits.py 12547
 *     ***    ****       *     *****
**    *   *   *          **         *
 *       *    *         * *        *
 *      *      ***     *  *       *
 *     *           *   ******     *
 *    *         *   *       *     *
***   *****     ***         *     *
```

BigDigits

Faça um script que leia do terminal, já na chamada do programa, um número escrito pelo usuário e imprima este número usando uma lista de strings para cada algarismo com `*` e espaços.

Exemplo de algarismo:

```
Dois = [" *** ",
        "*   *",
        "    * ",
        "   *  ",
        "  *   ",
        " *    ",
        "*     ",
        "*****"]
```

Você deve considerar duas exceções:

1. O usuário não digitou nenhum número na chamada.
2. O usuário digitou um caracter não numérico na chamada.

Utilize a biblioteca sys (inclua `import sys` logo abaixo da linha shebang) para obter o número digitado usando sys.argv[1].

Exemplo de entrada e saida:

```
louisaugusto@louisaugusto-Desktop:~$ python3 bigdigits.py 12547
 *     ***    ****      *     *****
**    *   *   *         **         *
 *       *    *        * *        *
 *      *      ***     *  *       *
 *     *           *   ******     *
 *    *        *   *      *     *
***   *****    ***       *     *
```

BigDigits

Faça um script que leia do terminal, já na chamada do programa, um número escrito pelo usuário e imprima este número usando uma lista de strings para cada algarismo com `*` e espaços.

Exemplo de algarismo:

```
Dois = [" *** ",
        "*   *",
        "    * ",
        "   *  ",
        "  *   ",
        "*     ",
        "*****"]
```

Você deve considerar duas exceções:

1. O usuário não digitou nenhum número na chamada.
2. O usuário digitou um caracter não numérico na chamada.

Utilize a biblioteca sys (inclua `import sys` logo abaixo da linha shebang) para obter o número digitado usando sys.argv[1].

Exemplo de entrada e saida:

```
louisaugusto@louisaugusto-Desktop:~$ python3 bigdigits.py 12547
 *     ***    ****      *     *****
**    *   *   *         **         *
 *       *    *        * *        *
 *      *      ***     *  *       *
 *     *           *   ******     *
 *    *        *   *      *     *
***   *****    ***       *     *
```

BigDigits

Faça um script que leia do terminal, já na chamada do programa, um número escrito pelo usuário e imprima este número usando uma lista de strings para cada algarismo com * e espaços.

Exemplo de algarismo:

```
Dois = [" *** ",
        "*   *",
        "    * ",
        "   *  ",
        "  *   ",
        "*     ",
        "*****"]
```

Você deve considerar duas exceções:

1. O usuário não digitou nenhum número na chamada.
2. O usuário digitou um caracter não numérico na chamada.

Utilize a biblioteca sys (inclua `import sys` logo abaixo da linha shebang) para obter o número digitado usando sys.argv[1].

Exemplo de entrada e saida:

```
louisaugusto@louisaugusto-Desktop:~$ python3 bigdigits.py 12547
 *     ***    ****      *     *****
**    *   *   *         **        *
 *       *   *        * *       *
 *      *      ***    *  *      *
 *     *           *   ******    *
 *    *        *   *      *     *
***   *****    ***       *     *
```

BigDigits

Faça um script que leia do terminal, já na chamada do programa, um número escrito pelo usuário e imprima este número usando uma lista de strings para cada algarismo com * e espaços.

Exemplo de algarismo:

```
Dois = [" *** ",
        "*   *",
        "    * ",
        "   *  ",
        "  *   ",
        "*     ",
        "*****"]
```

Você deve considerar duas exceções:

1. O usuário não digitou nenhum número na chamada.
2. O usuário digitou um caracter não numérico na chamada.

Utilize a biblioteca sys (inclua `import sys` logo abaixo da linha shebang) para obter o número digitado usando sys.argv[1].

Exemplo de entrada e saida:

```
louisaugusto@louisaugusto-Desktop:~$ python3 bigdigits.py 12547
 *     ***    ****      *     *****
**    *   *   *         **        *
 *       *   *        * *       *
 *      *      ***    *  *      *
 *     *           *   ******    *
 *    *        *   *      *     *
***   *****    ***       *     *
```

BigDigits

Faça um script que leia do terminal, já na chamada do programa, um número escrito pelo usuário e imprima este número usando uma lista de strings para cada algarismo com `*` e espaços.

Exemplo de algarismo:

```
Dois = [" *** ",
        "*   *",
        "    * ",
        "   *  ",
        "  *   ",
        "*     ",
        "*****"]
```

Você deve considerar duas exceções:

1. O usuário não digitou nenhum número na chamada.
2. O usuário digitou um caracter não numérico na chamada.

Utilize a biblioteca sys (inclua `import sys` logo abaixo da linha shebang) para obter o número digitado usando sys.argv[1].

Exemplo de entrada e saida:

```
louisaugusto@louisaugusto-Desktop:~$ python3 bigdigits.py 12547
 *     ***    ****      *     *****
**    *   *   *         **        *
 *       *   *        * *       *
 *      *      ***    *  *      *
 *     *           *   ******    *
 *    *       *   *      *     *
***   *****    ***       *     *
```

BigDigits

Faça um script que leia do terminal, já na chamada do programa, um número escrito pelo usuário e imprima este número usando uma lista de strings para cada algarismo com `*` e espaços.

Exemplo de algarismo:

```
Dois = [" *** ",
        "*   *",
        "    * ",
        "   *  ",
        "  *   ",
        "*     ",
        "*****"]
```

Você deve considerar duas exceções:

1. O usuário não digitou nenhum número na chamada.
2. O usuário digitou um caracter não numérico na chamada.

Utilize a biblioteca sys (inclua `import sys` logo abaixo da linha shebang) para obter o número digitado usando sys.argv[1].

Exemplo de entrada e saida:

```
louisaugusto@louisaugusto-Desktop:~$ python3 bigdigits.py 12547
 *     ***    ****       *     *****
**    *   *   *          **         *
 *       *   *         * *        *
 *      *      ***    *  *       *
 *     *           *   ******    *
 *    *        *   *      *     *
***   *****    ***        *     *
```

BigDigits

Faça um script que leia do terminal, já na chamada do programa, um número escrito pelo usuário e imprima este número usando uma lista de strings para cada algarismo com `*` e espaços.

Exemplo de algarismo:

```
Dois = [" *** ",
        "*   *",
        "    * ",
        "   *  ",
        "  *   ",
        " *    ",
        "*     ",
        "*****"]
```

Você deve considerar duas exceções:

1. O usuário não digitou nenhum número na chamada.
2. O usuário digitou um caracter não numérico na chamada.

Utilize a biblioteca sys (inclua `import sys` logo abaixo da linha shebang) para obter o número digitado usando sys.argv[1].

Exemplo de entrada e saida:

```
louisaugusto@louisaugusto-Desktop:~$ python3 bigdigits.py 12547
 *      ***     ****        *      *****
**     *   *    *           **          *
 *         *    *          * *          *
 *       *       ***      *  *         *
 *      *             *    ******     *
 *     *         *    *       *      *
***    *****     ***         *      *
```