CSIT881
Programming and Data Structures

Exception Handling

UNIVERSITY
OF WOLLONGONG
AUSTRALIA

Objectives

- Exception handling
- Catch an exception
- Raise an exception
- Create a new exception class

Why do we need exception handling?

Consider the following program:
what happens if the user doesn't enter integer number?

```
# ask user to enter an integer
user_input = input("Enter an integer: ")
number = int(user_input)

# display the integer
print("You have entered {0}".format(number))
```

```
Enter an integer: 10
You have entered 10
```

```
Enter an integer: abc
ValueError: invalid literal for int() with base 10: 'abc'
```

When exception is thrown, we need to handle it otherwise the program will crash.

Exception handling helps the program terminate gracefully.

Exception handling syntax

```
try:
   # the try block


except ExceptionA as e:
   # handle ExceptionA in this block


except ExceptionB as e:
   # handle ExceptionB in this block


except ExceptionC as e:
   # handle ExceptionC in this block

...
```

First, the **try** block is executed.

If no exception occurs, the except blocks are skipped and execution of the try statement is finished.

If an exception occurs then the rest try block is skipped.

The corresponding **except** block is executed.

If an exception occurs, but there is no exception type that matches then the program will crash.

Exception handling syntax

```
try:
   # the try block


except ExceptionA as e:
   # handle ExceptionA in this block


except ExceptionB as e:
   # handle ExceptionB in this block


except ExceptionC as e:
   # handle ExceptionC in this block

except:
   # handle any other kind of exceptions
```

This **except** clause is used to catch all type of unhandled exceptions.

This is optional.

Exception handling syntax

```
try:
   # the try block


except ExceptionA as e:
   # handle ExceptionA in this block


except ExceptionB as e:
   # handle ExceptionB in this block


except ExceptionC as e:
   # handle ExceptionC in this block

else:
   # if there is no exceptions
```

If there is no exception then this **else** block is executed.

This is optional.

Exception handling syntax

```
try:
    # the try block


except ExceptionA as e:
    # handle ExceptionA in this block


except ExceptionB as e:
    # handle ExceptionB in this block


except ExceptionC as e:
    # handle ExceptionC in this block

finally:
    # exceptions or no exception
    # this block always get executed
```

Exception or no exceptions, this **finally** block is always got executed last.

This is optional.

Common exception

ValueError

Raised when an operation or function receives an argument that has the right type but an inappropriate value

```
# ask user to enter an integer
user_input = input("Enter an integer: ")
number = int(user_input)
```

```
Enter an integer: abc
ValueError: invalid literal for int() with base 10: 'abc'
```

Common exception

ZeroDivisionError

Raised when the second argument of a division or modulo operation is zero.

```
# ask user to enter 2 integers
user_input = input("Enter the 1st integer: ")
number1 = int(user_input)

user_input = input("Enter the 2nd integer: ")
number2 = int(user_input)

# calculate the quotient
quotient = number1 / number2
```

```
Enter the 1st integer: 13
Enter the 2nd integer: 0
ZeroDivisionError: division by zero
```

Common exception

ModuleNotFoundError

Raised by import when a module could not be located.

```
import rand

number = rand.randint(1, 6)

print(number)
```

```
ModuleNotFoundError: No module named 'rand'
```

Common exception

NameError

Raised when a variable is not found.

```
while (cat < 10):
    print(cat)
    cat = cat + 1
```

```
NameError: name 'cat' is not defined
```

Common exception

IndexError

Raised when a sequence subscript is out of range.

```
sentence = "Hi there!"

letter = sentence[0]        OK
print(letter)

letter = sentence[10]       NOT OK
print(letter)
```

```
IndexError: string index out of range
```

Exception class hierarchy

$$\begin{tikzcd}
 & \texttt{BaseException} & \\
 & \texttt{Exception} \arrow[u] & \\
\texttt{ValueError} \arrow[ur] & \texttt{NameError} \arrow[u] & \texttt{ArithmeticError} \arrow[ul] \\
 & & \texttt{ZeroDivisionError} \arrow[u]
\end{tikzcd}$$

`Class BaseException`
The base class for all built-in exceptions.

`Class Exception`
All user-defined exceptions should be derived from this class.

Coding convention: User-defined exception class name should end with the word Error.

Example: `ValueError`, `ZeroDivisionError`, `ArithmeticError`, `MemoryError`, `ModuleNotFoundError`, etc.

Raise an exception

Sometimes we want to manually **raise an exception** to indicate an error has occurred.

Example: raise an exception with no error message

```
# ask user to enter a positive integer
user_input = input("Enter a positive integer: ")
number = int(user_input)

# if number is not positive then raise exception
if(number <= 0):
  raise ValueError
```

Example: raise an exception with an error message

```
# ask user to enter a positive integer
user_input = input("Enter a positive integer: ")
number = int(user_input)

# if number is not positive then raise exception
if(number <= 0):
  raise ValueError("Input must be a positive number ")
```

Create a new exception class

All user-defined exceptions should be derived from class `Exception`.

Coding convention: User-defined exception class name should ends with the word Error.

```
class BlahBlahBlahError(Exception):
#{
  """
  A new exception class inherit from Exception
  """

  def __init__(self, message):
  #{
    self.message = message
  #}

#}
```

Example 1: Integer input with ValueError

Get integer input with exception handling

```
try:
  # ask user to enter an integer
  user_input = input("Enter an integer: ")
  number = int(user_input)

  # display the integer
  print("You have entered {0}".format(number))

except ValueError as e:
  print(e)
```

```
Enter an integer: 10
You have entered 10
```

```
Enter an integer: abc
invalid literal for int() with base 10: 'abc'
```

Example 1: Integer input with ValueError

Rewrite the program using a user-friendly message

```
try:
  # ask user to enter an integer
  user_input = input("Enter an integer: ")
  number = int(user_input)

  # display the integer
  print("You have entered {0}".format(number))

except ValueError as e:
  print("You have entered an invalid number")
```

```
Enter an integer: 10
You have entered 10
```

```
Enter an integer: abc
You have entered an invalid number
```

```
Enter an integer: 1.7
You have entered an invalid number
```

Example 2: Decimal input with ValueError

```python
try:
  # ask user to enter a decimal number
  user_input = input("Enter a decimal number: ")
  number = float(user_input)

  # display the number
  print("You have entered {0}".format(number))

except ValueError as e:
  print("You have entered an invalid number")
```

```
Enter an integer: 2.3
You have entered 2.3
```

```
Enter an integer: 5
You have entered 5.0
```

```
Enter an integer: abc
You have entered an invalid number
```

Example 3: catching 2 types of exception

Consider the following program:

what happens if the user enter zero for the second number

```
try:
  # ask user to enter 2 integers
  user_input = input("Enter the 1st integer: ")
  number1 = int(user_input)

  user_input = input("Enter the 2nd integer: ")
  number2 = int(user_input)

  # calculate the quotient
  quotient = number1 / number2

  # display the division equation
  print("{0} / {1} = {2}".format(number1, number2, quotient))

except ValueError as e:
  print("You have entered an invalid number")
```

```
Enter the 1st integer: 13
Enter the 2nd integer: 2
13 / 2 = 6.5
```

```
Enter the 1st integer: 13
Enter the 2nd integer: abc
You have entered an invalid number
```

```
Enter the 1st integer: 13
Enter the 2nd integer: 0
ZeroDivisionError: division by zero
```

Example 3: catching 2 types of exception

Rewrite to catch another type of exception

```
try:
  # ask user to enter 2 integers
  user_input = input("Enter the 1st integer: ")
  number1 = int(user_input)

  user_input = input("Enter the 2nd integer: ")
  number2 = int(user_input)

  # calculate the quotient
  quotient = number1 / number2

  # display the division equation
  print("{0} / {1} = {2}".format(number1, number2, quotient))

except ValueError as e:
  print("You have entered an invalid number")

except ZeroDivisionError as e:
  print("Invalid division - cannot divide by 0")
```

```
Enter the 1st integer: 13
Enter the 2nd integer: abc
You have entered an invalid number
```

```
Enter the 1st integer: 13
Enter the 2nd integer: 0
Invalid division - cannot divide by 0
```

Example 4: Raise exception

```
try:
  # ask user to enter a positive integer
  user_input = input("Enter a positive integer: ")
  number = int(user_input)

  # display the number
  print("You have entered {0}".format(number))

except ValueError as e:
  print("You have entered an invalid number")
```

```
Enter a positive integer: 13
You have entered 13
```

```
Enter a positive integer: abc
You have entered an invalid number
```

What happens if the user enters zero or negative number?

```
Enter a positive integer: -13
You have entered -13
```

Example 4: Raise exception

Rewrite to raise exception when the number is not positive:

```
try:
  # ask user to enter a positive integer
  user_input = input("Enter a positive integer: ")
  number = int(user_input)

  # if number is not positive then raise exception
  if(number <= 0):
    raise ValueError

  # display the number
  print("You have entered {0}".format(number))

except ValueError as e:
  print("You have entered an invalid number")
```

```
Enter a positive integer: abc
You have entered an invalid number
```

```
Enter a positive integer: -13
You have entered an invalid number
```

Can we have different error message to distinguish these two cases?

```
Enter a positive integer: 13
You have entered 13
```

Example 4: Raise exception

```
try:
  # ask user to enter a positive integer
  user_input = input("Enter a positive integer: ")

  try:
    number = int(user_input)
  except:
    raise ValueError("Invalid integer format")

  # if number is not positive then raise exception
  if(number <= 0):
    raise ValueError("Input must be a positive number")

  # display the number
  print("You have entered {0}".format(number))

except ValueError as e:
  print("Error: " + str(e))
```

```
Enter a positive integer: abc
Error: Invalid integer format
```

```
Enter a positive integer: -13
Error: Input must be a positive number
```

Example 5: Keep asking user until a valid input is entered

Write a program to ask the user to enter a positive integer.

The program should keep asking until the user enters a valid input.

```
Enter a positive integer:  abc
Error: Invalid integer format

Enter a positive integer:  -10
Error: Input must be a positive number

Enter a positive integer:  xyz
Error: Invalid integer format

Enter a positive integer:  0
Error: Input must be a positive number

Enter a positive integer:  5
You have entered 5
```

```
Enter a positive integer:  38
You have entered 38
```

Example 5

The program should keep asking until the user enters a valid input.

```
# keep asking until we got a valid number
while True:
#{
  try:
    # ask user to enter a positive integer
    user_input = input("Enter a positive integer: ")

    # attempt to translate input into integer
    try:
      number = int(user_input)
    except:
      raise ValueError(" Invalid integer format ")

    # if number is not positive then raise exception
    if(number <= 0):
      raise ValueError(" Input must be a positive number ")

    # if we get to here- then we should have a valid number
    print("You have entered {0}".format(number))
    break

  except ValueError as e:
    print("Error: " + str(e))
#}
```

Example 6: create a new exception class

```
Breakfast menu at Whosville Eatery
1) Green eggs and ham
2) Red breads with jam
3) Blue salad with lamb chops
Enter your selection (1/2/3): 5
You have to choose 1, 2 or 3.
```

```
Breakfast menu at Whosville Eatery
1) Green eggs and ham
2) Red breads with jam
3) Blue salad with lamb chops
Enter your selection (1/2/3): 1

Drink size:
S) Small
M) Medium
L) Large
Enter your selection (S/M/L): K
You have to choose S, M or L.
```

```
Breakfast menu at Whosville Eatery
1) Green eggs and ham
2) Red breads with jam
3) Blue salad with lamb chops
Enter your selection (1/2/3): 2

Drink size:
S) Small
M) Medium
L) Large
Enter your selection (S/M/L): M

Your order is red breads with jam
and a medium drink
```

We will create a new exception class called `BadInputError` and raise this exception whenever the user enters an invalid input.

Example 6

We will create a new exception class called `BadInputError` and raise this exception whenever the user enters an invalid input.

```
class BadInputError(Exception):
  """
  An exception class when user enters a wrong input with attribute
  message: the error message
  """

  def __init__(self, message):
    self.message = message
```

```
try:

  # food order

  # drink order

  # display order

except BadInputError as e:

  # display error message
  print(e.message)
```

Example 6

```
try:
  # food order
  print("1) Green eggs and ham")
  print("2) Red breads with jam")
  print("3) Blue salad with lamb chops")

  food_option = input("Enter your selection (1/2/3): ")

  if food_option == "1":
    food = "green eggs and ham"
  elif food_option == "2":
    food = "red breads with jam"
  elif food_option == "3":
    food = "blue salad with lamb chops"
  else:
    raise BadInputError("You have to choose 1, 2 or 3.")

  # drink order

  # display order

except BadInputError as e:

  # display error message
  print(e.message)
```

Example 6

```
try:
  # food order
  ...

  # drink order
  print("Drink size:")
  print("S) Small")
  print("M) Medium")
  print("L) Large")

  drink_option = input("Enter your selection (S/M/L): ")

  if drink_option == "S":
    drink = "small drink"
  elif drink_option == "M":
    drink = "medium drink"
  elif drink_option == "L":
    drink = "large drink"
  else:
    raise BadInputError("You have to choose S, M or L.")

  # display order

except BadInputError as e:

  # display error message
  print(e.message)
```

Example 6

```
try:
   # food order
   ...

   # drink order
   ...

   # display order
  print("Your order is {0} and a {1}".format(food, drink))

except BadInputError as e:

   # display error message
  print(e.message)
```

Test your coding skill

Write a program to ask the user to enter a first name, a last name and an email. When there is an input error, the program has to stop and display appropriate error.

Below are possible errors:

- First name is empty
- Last name is empty
- Email in wrong format

```
Enter first name:
Error: First name must not be empty
```

```
Enter first name: John
Enter last name:
Error: Last name must not be empty
```

```
Enter first name: John
Enter last name: Smith
Enter email: blah
Error: Invalid email
```

```
Enter first name: Green
Enter last name: Frog
Enter email: frog@pond.com
Thank you for your input
```