

CSIT110

Fundamental Programming with Python

Exception Handling

Goh X. Y.



In this lecture

- Exception handling
- Catch an exception
- Raise an exception
- Creating new exception class

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(f"You have entered {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

Rewrite the program using try-except

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

    # display the integer
    print(f"You have entered {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'
```

Exception handling

Rewrite the program using try-except and 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(f"You have entered {number}")

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

Enter an integer: **abc**
You have entered an invalid integer

Enter an integer: **1.7**
You have entered an invalid integer

Exception handling

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

    # display the number
    print(f"You have entered {number}")

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

Enter a decimal number: **2.3**
You have entered 2.3

Enter a decimal number: **5**
You have entered 5.0

Enter a decimal number: **abc**
You have entered an invalid number

Exception handling

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(f"{number1} / {number2} = {quotient}")

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

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

Exception handling

Rewrite to catch another type of exception

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

```
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(f"{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")
```


Exception handling

```
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 an exception occurs then the rest of the try block is skipped. The corresponding **except** block is executed.

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

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

Exception handling

```
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 other types of unhandled exceptions

Exception handling

```
try:
    # the try block

except ExceptionA as e:
    # handle ExceptionA in this block

except ExceptionB as e:
    # handle ExceptionB in this block

except:
    # handle any other kind of exceptions

else:
    # executed if no exceptions was raised
```

This **else** clause is used for instructions if no exception was raised in the try block

Exception handling

```
try:
    # the try block

except ExceptionA as e:
    # handle ExceptionA in this block

except:
    # handle any other kind of exceptions

else:
    # executed if no exceptions was raised

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

← Exception or no exceptions, this **finally** block will always be executed after the other blocks

Raising an Exception

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

    # display the number
    print(f"You have entered {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

Raising an Exception

```
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(f"You have entered {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

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

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

Raising an Exception

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

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

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

Enter a positive integer: **abc**
Error: 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(f"You have entered {number}")
```

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

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

Exception - Example

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
```


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

```
valid_number_flag = False # have we got a valid number?
while (not valid_number_flag): # keep asking until we got a valid number
    try:
        # ask user to enter a positive integer
        user_input = input("Enter a positive integer: ")
        # attempt to convert input into an integer
        try:
            number = int(user_input)
        except:
            raise ValueError("Invalid integer format")

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

        # if we get to here- then we should have a valid number
        valid_number_flag = True

        print(f"You have entered {number}")
    except ValueError as e:
        print("Error: " + str(e))
```

Challenge yourself!

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.

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
```

Creating an exception class

Class BaseException

The base class for all built-in exceptions.

Class Exception

All user-defined exceptions should inherit this class.

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

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

Example context

```
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.
```

```
Enter your selection (1/2/3): 5
You have to choose 1, 2 or 3.
```

```
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.
```

```
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.

Creating an Exception Class - Example

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:  
    # get food order  
    # get drink order  
    # display order  
except BadInputError as e:  
    # display error message  
    print(e.message)
```

Raising an Exception - Example

```
try:
    # get 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.")

    # get drink order

    # display order

except BadInputError as e:
    print(e.message) # display error message
```

Raising an Exception - Example

```
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 ValueError("You have to choose S, M or L.")
    # display order
except ValueError as e:
    # display error message print(e.message)
```

Example

```
try:
    # food order
    ...
    # drink order
    ...
    # display order
    print(f"Your order is {food} and a {drink}")

except KeyboardInterrupt as e:
    # display error message print(e.message)
```


Any questions?