Exception Handling in Python

Created by :- The Easylearn Academy
9662512857

What is exception in python

- Exception means run time error in python program. Exception are type of error which may or may not occur during execution of program.
- Exception depends upon user input or environment in which program runs.
- For example in program which divides one from another number given by user, if user give correct give both number correctly then error will not come and program will finish sucessfully
- But if one or both number are not valid then program will suddenly stop (which is called crash) now such situation is called exception.
- If any type of exception occur in program then program can not complete task for which it is developed.
- So must develop program in way that if exception occur in program then program should recover itself from it and try to finish given task.
- Python has many built-in exceptions which forces your program to output an error when something in it goes wrong.

Catching Exceptions in Python

- In Python, exceptions can be handled using a try statement.
- A critical operation which can raise exception is placed inside the try clause and the code that handles exception is written in except clause.

It is up to us, what operations we perform once we have caught the exception.

Syntax

Here is simple syntax of try....except...else blocks −

```
try:
   You do your operations here;
   ......................
except ExceptionI:
   If there is ExceptionI, then execute this block.
except ExceptionII:
   If there is ExceptionII, then execute this block.
   ......................
else:
   If there is no exception then execute this block.
```

Important points

- A single try statement can have multiple except statements. This is useful when the try block contains statements that may throw different types of exceptions.
- You can also provide a generic except clause, which handles any exception.
- After the except clause(s), you can include an else-clause. The code in the else-block executes if the code in the try: block does not raise an exception.
- The else-block is a good place for code that does not need the try: block's protection.

Example

exception handling 1

```
#write a program to findout & display qube of positive given number
try:
    number = int(input("Enter positive number to findout its qube"))
    if number<0: #< > <= >= == !=
        print("you have entered negative number")
        number = 0 - number # 0 - -10
    qube = number * number * number # 10 * 10
    print(f"qube = {qube}")
except ValueError:
    print("invalid input only numbers are allowed")
```

example of user defined exception with class

```
#make sum & average of all numbers in list
numbers = ['test',10,20,30,40,50,100,None]
sum = 0
count = 0
for item in numbers:
    print(item)
    try:
        sum = sum + item #if error occured line no 9 will not run
        count = count + 1
    except:
        print("oops something happens, but i will manage")

average = sum/count
print(f"sum = {sum} count = {count}")
print(f"average = {average}")
```

Example of simple user defined exception

exception handling 1

```
#example of user defined exception
'''
    write a program to accept age from user if age<18 or age>60 raise custom exception with
message unusal age for job
'''
try:
    age = int(input("Enter your age at the time of starting first day of job"))
    if age<18 or age>60:
        raise ValueError
    else:
        remaining_job_year = 60 - age
        print(f"you have {remaining_job_year} left for job")
except ValueError:
    print("unusal age for job")

print("Good bye")
```

The *except* Clause with No Exceptions

```
try:
    You do your operations here;
    ......................
except:
    If there is any exception, then execute this block.
    ......................
else:
    If there is no exception then execute this block.
```

- This kind of a **try-except** statement catches all the exceptions that occur.
- Using this kind of try-except statement is not considered a good programming practice though, because it catches all exceptions but does not make the programmer identify the root cause of the problem that may occur.

The *except* Clause with Multiple Exceptions

```
you can also use the same except statement to handle multiple
   exceptions as follows –

try:
   You do your operations here;
   ......................
except(Exception1[, Exception2[,...ExceptionN]]]):
   If there is any exception from the given exception list,
   then execute this block.
   ......................
else:
   If there is no exception then execute this block.
```

example of user defined exception with class

```
try:
    # Take inputs
    num1 = int(input("Enter numerator: "))
    num2 = int(input("Enter denominator: "))

    # Perform division
    result = num1 / num2
    print("Result:", result)

except ValueError:
    print("Invalid input! Please enter numeric values only.")

except ZeroDivisionError:
    print("Denominator cannot be zero.")

except Exception as e:
    print("Something went wrong:", e)
```

The try-finally Clause

- You can use a finally: block along with a try: block. The finally block is a place to put any code that must execute, whether the try-block raised an exception or not. The syntax of the try-finally statement is this −

```
try:
   You do your operations here;
   ......................
   Due to any exception, this may be skipped.
finally:
   This would always be executed.
   ......................
```

try catch and finally

```python
from datetime import datetime

try:
    user_input = input("Enter your birth date (DD-MM-YYYY): ")
    birth_date = datetime.strptime(user_input, "%d-%m-%Y")
    print("Your birth date is:", birth_date.strftime("%A, %d %B %Y"))

except ValueError:
    print("Invalid date format! Please enter in DD-MM-YYYY format.")

finally:
    print("Thank you for using the date checker.")
```

Else

- You can use the else keyword to define a block of code to be executed if no errors were raised:
- Example

try catch and else & finally

```
from datetime import datetime
try:
    user_input = input("Enter your appointment date (DD-MM-YYYY): ")
    appointment_date = datetime.strptime(user_input, "%d-%m-%Y")

    if appointment_date < datetime.today():
        raise ValueError("The date must be in the future.")

except ValueError as e:
    print("Error:", e)
else:
    print("Your appointment is scheduled on:", appointment_date.strftime("%A, %d %B %Y"))
finally:
    print("Thank you for using the appointment scheduler.")
```

Raising an Exceptions

- In Python programming, exceptions are raised when corresponding errors occur at run time, but we can forcefully raise it using the keyword raise.
- We can also optionally pass in value to the exception to clarify why that exception was raised.

```
try:
        age = int(input("Enter the age?"))
        if age<18:
            raise ValueError;
        else:
            print("the age is valid")
    except ValueError:
        print("The age is not valid")
```

Custom Exception / user defined exception

- python also allows you to create your own exceptions by deriving classes from the standard built-in exceptions.
- In Python, users can define such exceptions by creating a new class. This exception class has to be derived, either directly or indirectly, from Exception class. Most of the built-in exceptions are also derived form this class.
- We will learn this topic after we learn object and classes in python

example of user defined exception with class

```
#example of user defined exception (custom exception)
#write a program to findout qube of positive number, if user give negative number raise custom exception NegativeNumber which will convert negative number into postive number
class NegativeNumberException(Exception):
    def __init__(self,message):
        super().__init__(message)
        self.message = message
    def getMessage(self):
        return self.message
while True:
    try:
        number = int(input("Enter postive number"))
        while True:
            try:
                if number<0: #negative number
                    #raise custom exception NegativeNumberException
                    raise NegativeNumberException("you can not pass negative number")
                else:
                    qube = number * number * number
                    print(f"qube = {qube}")
                    break #break from innerloop
            except NegativeNumberException as error:
                print(error.getMessage())
                number = 0 - number
        break #break from outer loop
    except ValueError:
        print('it is invalid value')
```