PROGRAMMING WITH PYTHON

Karuna sheel

NIELIT kurukshetra

01

An exception can be defined as an abnormal condition in a program resulting in the disruption in the flow of the program.

02

Whenever an exception occurs, the program halts the execution, and thus the further code is not executed. Therefore, an exception is the error which python script is unable to tackle with.

03

Python provides us with the way to handle the Exception so that the other part of the code can be executed without any disruption. However, if we do not handle the exception, the interpreter doesn't execute all the code that exists after the that.

Python Exceptions

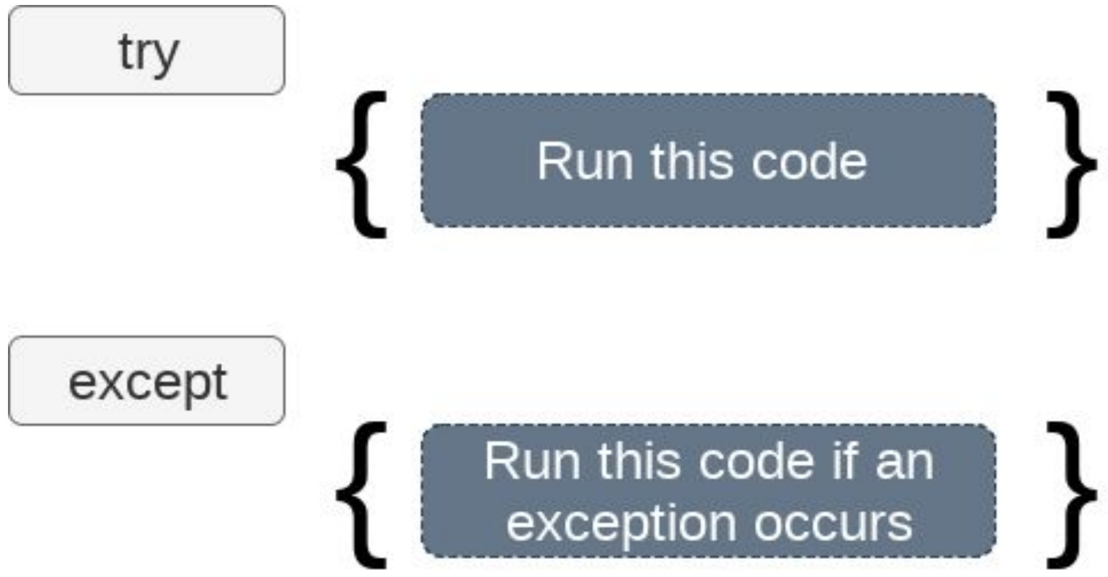
Common Exceptions

A list of common exceptions that can be thrown from a normal python program is given below.

- **ZeroDivisionError**: Occurs when a number is divided by zero.
- **NameError**: It occurs when a name is not found. It may be local or global.
- **IndentationError**: If incorrect indentation is given.
- **IOError**: It occurs when Input Output operation fails.
- **EOFError**: It occurs when the end of the file is reached, and yet operations are being performed.

Exception Handling In Python

If the python program contains suspicious code that may throw the exception, we must place that code in the try block. The try block must be followed with the except statement which contains a block of code that will be executed if there is some exception in the try block.



Syntax

```
try:
#block of code
except Exception1:
#block of code
except Exception2:
#block of code
#other code
```

Problem without handling exceptions

Example

```
a = int(input("Enter a:"))
b = int(input("Enter b:"))
c = a/b;
print("a/b = "c)

#other code:
print("Hi I am other part of the program")
```

Output:

```
Enter a:10

Enter b:0

Traceback (most recent call last):

  File "exception-test.py", line 3, in <module>

    c = a/b;

ZeroDivisionError: division by zero
```

Example

```
# import module sys to get the type of exception

import sys
randomList = ['a', 0, 2]

for entry in randomList:
    try:
        print("The entry is", entry)
        r = 1/int(entry)
        break
    except:
        print("Oops!", sys.exc_info()[0], "occurred.")
        print("Next entry.")
        print()
print("The reciprocal of", entry, "is", r)
```

```
The entry is a
Oops! <class 'ValueError'>
occurred.
Next entry.

The entry is 0
Oops! <class 'ZeroDivisionError'>
occured.
Next entry.

The entry is 2
The reciprocal of 2 is 0.5
```

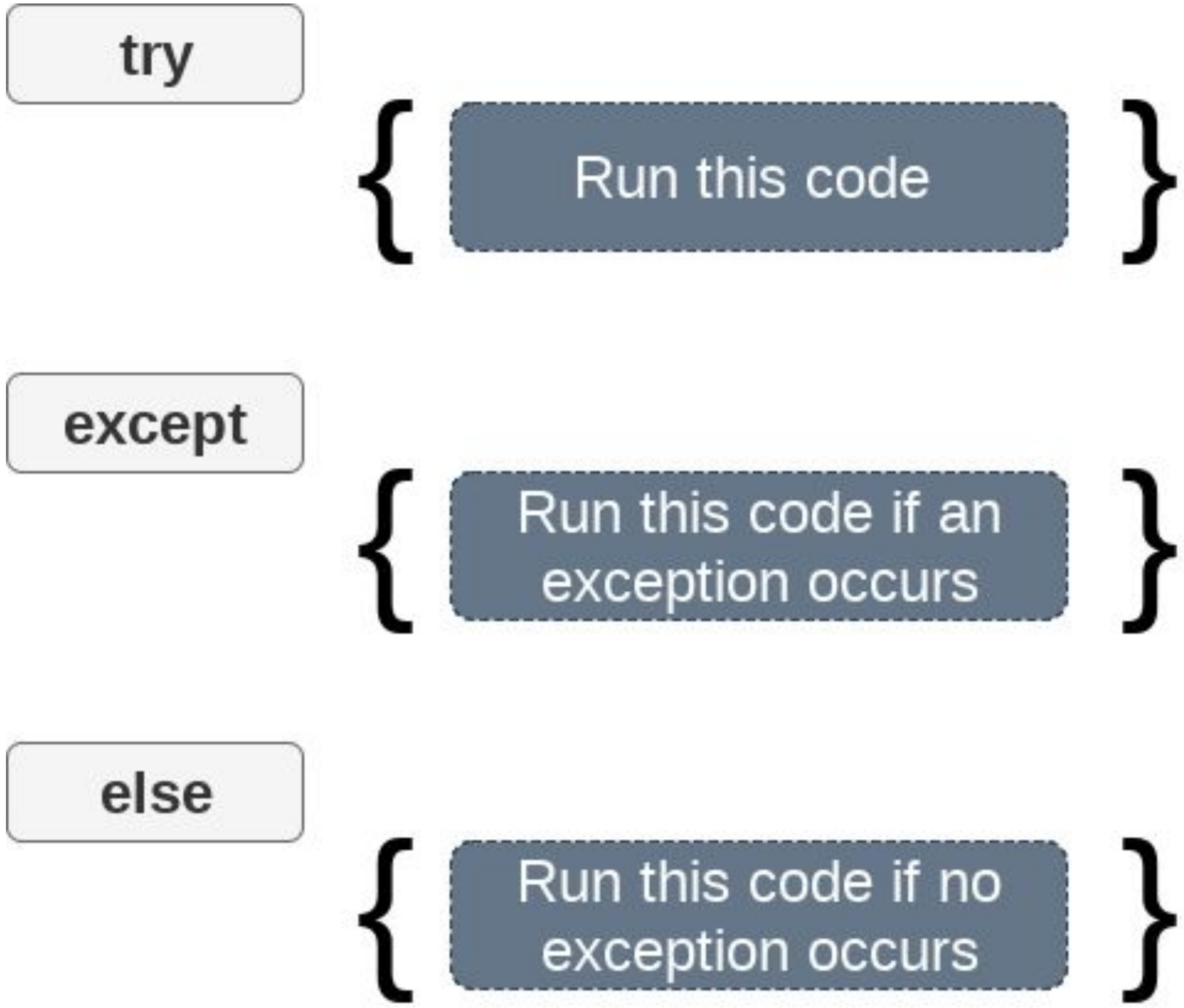
Syntax for Else Statement

We can also use the else statement with the try-except statement in which, we can place the code which will be executed in the scenario if no exception occurs in the try block.

The syntax to use the else statement with the try-except statement is given below.

```
try:
#block of code
except Exception1:
#block of code
else:
#this code executes if no except block is executed
```

Syntax


try
Run this code
except
Run this code if an exception occurs
else
Run this code if no exception occurs

Example

```
try:
    a = int(input("Enter a:"))
    b = int(input("Enter b:"))
    c = a/b;
    print("a/b = %d"%c)
except Exception:
    print("can't divide by zero")
else:
    print("Hi I am else block")
```

Output:

```
Enter a:10
Enter b:2
a/b = 5
Hi I am else block
```

The except statement with no exception

Python provides the flexibility not to specify the name of exception with the except statement.

Consider the following example.

Example

```
try:
    a = int(input("Enter a:"))
    b = int(input("Enter b:"))
    c = a/b;
    print("a/b = %d"%c)
except:
    print("can't divide by zero")
else:
    print("Hi I am else block")
```

Output:

```
Enter a:10
Enter b:0
can't divide by zero
```

Points to remember

- Python facilitates us to not specify the exception with the except statement.
- We can declare multiple exceptions in the except statement since the try block may contain the statements which throw the different type of exceptions.
- We can also specify an else block along with the try-except statement which will be executed if no exception is raised in the try block.
- The statements that don't throw the exception should be placed inside the else block.

Example

```
try:
    #this will throw an exception if the file doesn't exist.
    fileptr = open("file.txt","r")
except IOError:
    print("File not found")
else:
    print("The file opened successfully")
    fileptr.close()
```

Output:

```
File not found
```

Declaring multiple exceptions

The python allows us to declare the multiple exceptions with the except clause. Declaring multiple exceptions is useful in the cases where a try block throws multiple exceptions.

Syntax

```
try:
    #block of code

except (<Exception 1>,<Exception 2>,<Exception 3>,...<Exception n>)
    #block of code

else:
    #block of code
```

Example

```
try:
   a=10/0;
except (ArithmeticError,StandardError):
   print "Arithmetic Exception"
else:
   print "Successfully Done"
```

Output:

```
Arithmetic Exception
```

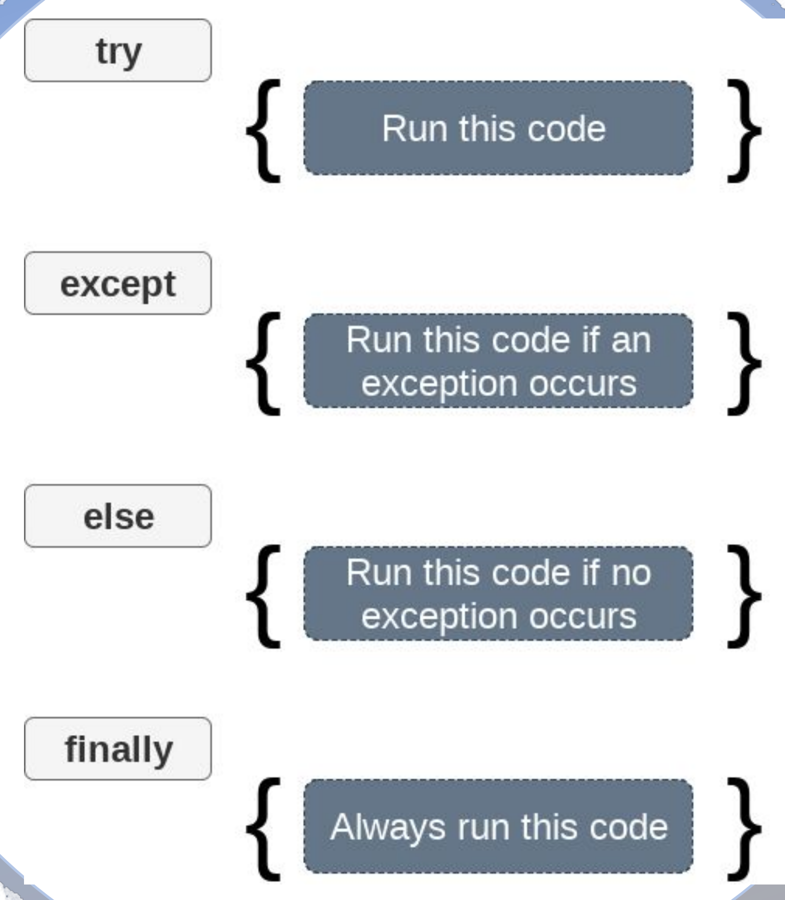
The finally block

We can use the finally block with the try block in which, we can pace the important code which must be executed before the try statement throws an exception.

The syntax to use the finally block is given below.

syntax

```
try:
    # block of code
    # this may throw an exception
finally:
    # block of code
    # this will always be executed
```


try
{ Run this code }
except
{ Run this code if an exception occurs }
else
{ Run this code if no exception occurs }
finally
{ Always run this code }

Example

```
try:
    fileptr = open("file.txt","r")
    try:
        fileptr.write("Hi I am good")
    finally:
        fileptr.close()
        print("file closed")
except:
    print("Error")
```

Output:

```
file closed
Error
```

Raising exceptions

The raise statement allows the programmer to force a specific exception to occur. The sole argument in raise indicates the exception to be raised. This must be either an exception instance or an exception class (a class that derives from Exception).

An exception can be raised by using the raise clause in python. The syntax to use the raise statement is given below.

syntax

```
raise Exception_class,<value>
```

Eg:

```
x = "hello"

if not type(x) is int:
  raise TypeError("Only integers are allowed")
```

Points to remember

- To raise an exception, raise statement is used. The exception class name follows it.
- An exception can be provided with a value that can be given in the parenthesis.
- To access the value "as" keyword is used. "e" is used as a reference variable which stores the value of the exception.

Example

```
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")
```

Output:

```
Enter the age?17
The age is not valid
```

Example

```
try:
a = int(input("Enter a?"))
b = int(input("Enter b?"))
if b is 0:
raise ArithmeticError;
else:
print("a/b = ",a/b)
except ArithmeticError:
print("The value of b can't be 0")
```

Output:

```
Enter a?10
Enter b?0
The value of b can't be 0
```

Custom Exception

The python allows us to create our exceptions that can be raised from the program and caught using the except clause. However, we suggest you read this section after visiting the Python object and classes.

Consider the following example.

Example

```
class ErrorInCode(Exception):
    def __init__(self, data):
        self.data = data
    def __str__(self):
        return repr(self.data)

try:
    raise ErrorInCode(2000)
except ErrorInCode as ae:
    print("Received error:", ae.data)
```

Output:

```
Received error: 2000
```