Exceptions ¶

What is Exception?

Exception is an event happened during code execution. It is most likely due to incorrect code or input.

Once an exception occurs, the code execution will stop if the exception is not handled.

Example:

Following code `result = 100 / 0` causes a `ZeroDivisionError` , and statement(s) following that line is not executed.

In [ ]:
```
# result = 100 / 0
# print('This line will not be executed')
```

Common Exception Types

Python provides several built-in Exceptions. You can also define your own Exception subclass.

Different exception are raised for diffent reasons.

Syntax Error

In [ ]:
```
# x = int('123'
```

Module Not Found

The `ModuleNotFoundError` occurs when an import statement fails.

In [ ]:
```
# import abc123
```

Unkown Object

The `NameError` will occur if an unknown variable/object is used, i.e. using an object before creating it.

In [ ]:
```
# print(my_invisible_var)
```

Index Out of Range

The `IndexError` occurs if you try to access an item by index which is outside the range of the list.

In [ ]:
```
# arr = [1,2,3]
# arr[3]
```

Wrong Data Type

The `TypeError` occurs when a function is called on a value of an inappropriate type.

In [ ]:
```
# x = 'a' + 2
```

ValueError

The `ValueError` occurs when a function is called on a value of the correct type, but with an inappropriate value.

For example, `int()` function is expecting its input to be a string of numerical type.

In [ ]:
```
# x = int('a')
```

AttributeError

Functions and variables of an object are collectly called `attributes` . When you try to access a non-existent attribute, e.g. due to typo mistake, an AttributeError will occur.

In [ ]:
```
# s = list(range(9))
# s.sort1()
```

Question:

How to you find the list of Error classes in Python, e.g. you forgot how to spell a type of Error?

In [ ]:
```

```

Exception Handling

Handling of exception in important and common in Python code. This is because python believes in `"Ask for forgiveness not permission"` .

To handle exceptions, use a `try/except` statement.

- The `try` block contains code that might throw an exception.

- If that exception occurs, the remaining code in the `try` block will be **skipped**, and the code in the `except` block is run.
- If no error occurs, the code in the `except` block doesn't run.

Exercise:

Use `try-except` to make sure `result = 100/0` statement doesn't cause program to fail.

- Print out `No division by zero` when exception occurs.

In [ ]:
```
try:
#     result = 100/0
    print('Result = ', result)
except D:
    print('No division by zero')
```

Try Code:

Run following code; Change `x = int('a')` to `x = int('999')` and try again.

```
try:
    print('point 1')
    x = int('a')
#    x = int('999')
    print("point 2")
except ValueError:
    print("point 3")
```

In [ ]:
```
try:
    print('point 1')
    x = int('a')
#    x = int('999')
    print("point 2")
except ValueError:
    print("point 3")
```

An `except` statement without any exception specified will catch all errors.

- A bad practice as it may catch unexpected errors and hide programming mistakes.

Try Code:

```
try:
    x = 10/ 0
except:
    print('An error occurred')
```

In [ ]:
```
try:
    x = 10/ 0
except:
    print('An error occurred')
```

Multiple Exceptions

A `try` statement can have multiple different `except` blocks to handle different exceptions.

- Multiple exceptions can also be put into a single except block using parentheses, to have the except block handle all of them.

Try Code:

Run following code. Comment/uncomment `y` assignment statement(s) and try again to see different exceptions.

```
try:
    x = 10
#   y = int('abc')
#   y = x / 0
    y = x + "hello")
    print('No exception')
except ZeroDivisionError:
    print("Divided by zero")
except (ValueError, TypeError):
    print("ValueError or TypeError occurred")
```

In [ ]:
```
try:
    x = 10
#   y = int('abc')
#   y = x / 0
    y = x + "hello"
    print('No exception')
except ZeroDivisionError:
    print("Divided by zero")
except (ValueError, TypeError):
    print("ValueError or TypeError occurred")
```

Print Error Message

You can print out the exception object, e.g. in log file, to find out more information about the error.

Try Code: Run following code; Comment/uncomment `y` assignment statement(s) to see different exceptions.

```
    try:
        x = 10
    #   y = int('abc')
    #   y = x / 0
        y = x + "hello"
        print('No exception')
    except Exception as e:
        print(e)
        print(repr(e))
```

In [ ]:
```
try:
    x = 10
#   y = int('abc')
#   y = x / 0
    y = x + "hello"
    print('No exception')
except Exception as e:
    print(e)
    print(repr(e))
```

Traceback

The `traceback` module provides methods for formatting and printing exceptions and their calling stacks, which is helpful in identifying the cause of error.

Try Code:

```
    import traceback

    try:
        x = 10
    #   y = int('abc')
    #   y = x / 0
        y = x + "hello"
        print('No exception')
    except Exception:
        traceback.print_exc()
```

In [ ]:
```
import traceback

try:
    x = 10
#   y = int('abc')
#   y = x / 0
    y = x + "hello"
    print('No exception')
except Exception:
    traceback.print_exc()
```

Else and Finally

Except

The `except` code block is placed after `try/except` statements.

If error does not occur, i.e. `except` code block is not executed, `else` code block will run.

Finally

The `finally` code block is placed at the bottom of a `try/except/else` statement.

Code within a `finally` statement always runs regardless whether an exception happens.

This is good place to put some code which always need to run, e.g. clean up or release resource.

Try Code:

Comment/uncomment statement `print(1 / 0)` and examine the printouts.

```
try:
    print("try")
#     print(1 / 0)
except ZeroDivisionError:    # execute if exeption
    print("except")
else:     # execute if no exception
    print("else")
finally:  # always execute
    print("finally")
```

In [ ]:
```
try:
    print("try")
#     print(1 / 0)
except ZeroDivisionError:    # execute if exeption
    print("except")
else:     # execute if no exception
    print("else")
finally:  # always execute
    print("finally")
```

Example

Close file regardless whether file operation is successful or not.

Try Code:

When a file is open in binary mode, its `write()` function expects content of binary data type, else a `TypeError` will occur.

```
try:
    f = open('abc', 'wb')
    f.write('abc')    # Error occurs
except Exception as e:
    print(repr(e))
finally:
    print('Close file')
    f.close()
```

In [ ]:
```
try:
    f = open('abc', 'wb')
    f.write('abc')    # Error occurs
except Exception as e:
    print(repr(e))
finally:
    print('Close file')
    f.close()
```