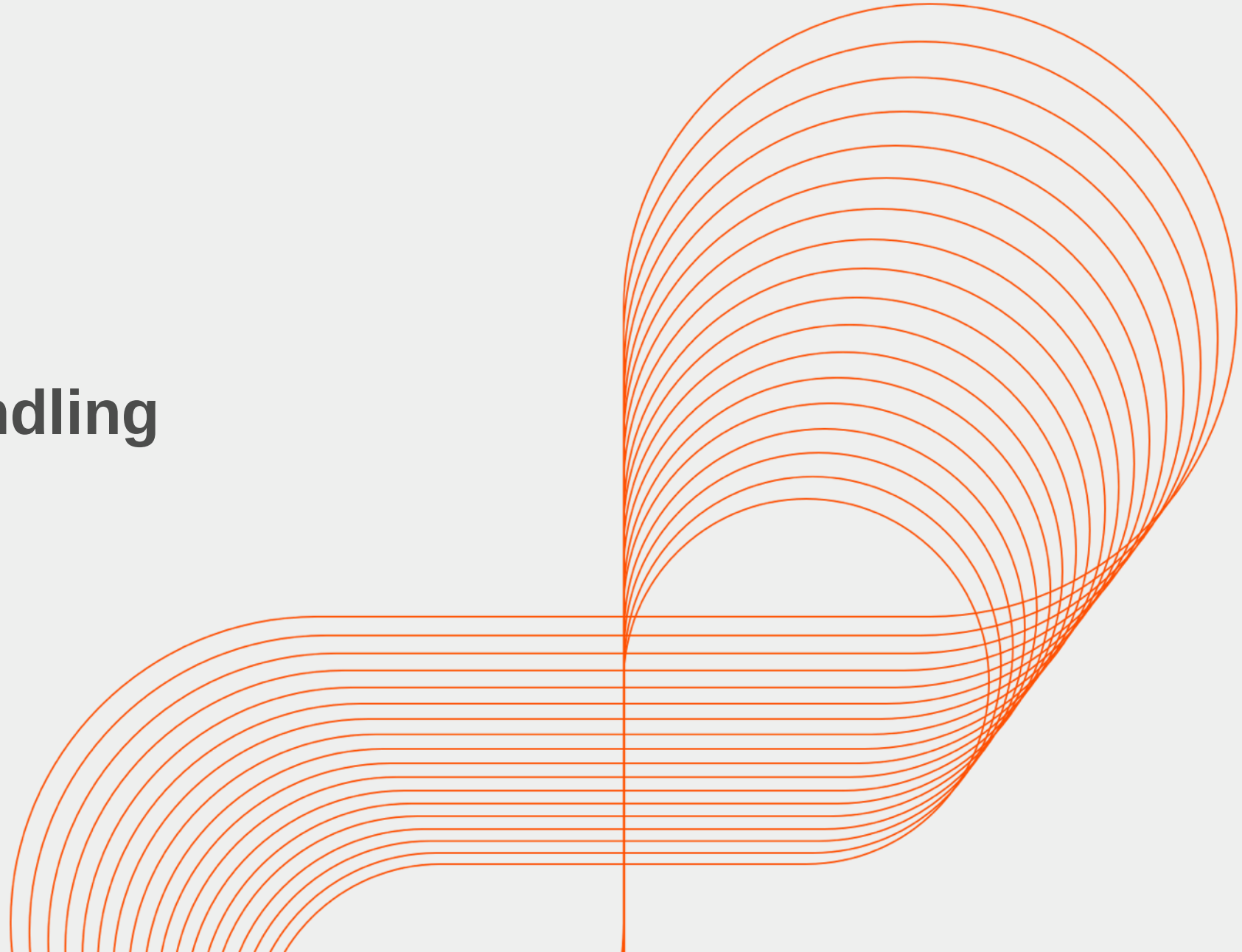
Persistent

# Exception Handling

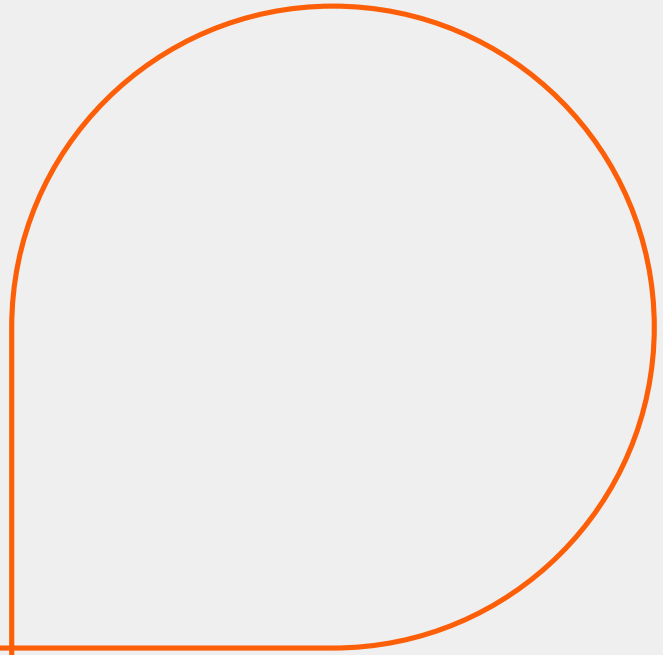


# Objectives

At the end of this session, you will be able to understand:

- Python Exception Handling

# Exceptions



## Python exceptions

- Languages like Python which support the raising and more importantly the handling of exceptions empowers the developer by placing them in a more direct line of control when errors occur
- The programmer not only has the ability to detect errors, but also to take more concrete and remedial actions when they occur. Due to the ability to manage errors during run-time, application robustness is increased
- Some of the examples shown in the earlier presentations of this course showed what happens when a Python program "crashes" or terminates due to unresolved errors
- Python provides a list of standard exceptions that are loaded into the interpreter as a built-in
- All standard/built-in exceptions are derived from the root class exception

## Standard exceptions

| Exception Name | Description |
|---|---|
| Exception | root class for all exceptions |
| SystemExit | request termination of Python interpreter |
| StandardError | base class for all standard built-in exceptions |
| ArithmeticError | base class for all numeric calculation errors |
| FloatingPointError | error in floating point calculation |
| OverflowError | calculation exceeded maximum limit for numerical type |
| EOFError | end-of-file marker reached without input from built-in |
| IOError | failure of input/output operation |

## Standard exceptions

| Exception Name | Description |
|---|---|
| ImportError | failure to import module or object |
| IndexError | no such index in sequence |
| KeyError | no such key in mapping |
| MemoryError | out-of-memory error (non-fatal to Python interpreter) |
| NameError | undeclared/uninitialized object (non-attribute) |
| RuntimeError | generic default error during execution |
| SyntaxError | error in Python syntax |
| ValueError | invalid argument given |

## Try-except statement

- The try-except statement allows to define a section of code to monitor for exceptions and also provides the mechanism to execute handlers for exceptions
- The syntax for the most general try-except statement looks like this:

```
try:
    # do stuff
except Exception:
    # handle exceptions
```

**Example:**

```
>>> try:
...         f = open('nonexistentfile')
... except IOError:
...         print ('could not open file')
```

## Try statement with multiple excepts

- Following is the general syntax for a try statement with multiple excepts.

```
try:
    # do stuff
except Exception1:
    # handle exception 1
except Exception 2:
    # handle exception 2
except Exception N:
    # handle exception N
```

## Try-finally statement

- The try-finally statement differs from try-except in that it is not used to handling exceptions
- Instead it is used to maintain consistent behavior regardless of whether or not exceptions occur
- The finally suite executes regardless of an exception being triggered within the try suite
- The general syntax of the try-finally statement is as shown below

```
try:
    try_suite
finally:
    finally_suite # executes regardless of exceptions
```

## Raising exceptions

- Exceptions may also be raised explicitly in your own code, through the use of the raise statement. The most basic form of this statement is

  raise exception(args)

- After the exception has been created, raise takes it and throws it upward along the stack of Python functions that were invoked in getting to the line containing the raise statement
- The new exception is thrown up to the nearest (on the stack) exception catcher looking for that type of exception
- If no catcher is found on the way to the top level of the program, this will either cause the program to terminate with an error or, in an interactive session, cause an error message to be printed to the console

## Defining new exceptions

- Python allows a developer to easily define new exceptions as shown below

```
class MyError(Exception):
    pass
```

- The above syntax creates a class that inherits everything from the base Exception class
- Python allows to raise, catch, and handle it like any other exception

- When given a single argument without catching and handling it, this will be printed at the end of the traceback as shown below:

```
>>> raise MyError("Some information about what went wrong")
Traceback (most recent call last):
    File "<stdin>", line 1, in <module>
__main__.MyError: Some information about what went wrong
```

- This argument will, of course, be available to a handler also as shown in the following slide

## Defining new exceptions (contd.)

- This argument to the user defined exception created in the earlier slide will, of course, be available to a handler also as shown below:

```
try:
    raise MyError("Some information about what went wrong")
except MyError as error:
    print("Situation:", error)
```

## Summary:

With this we have come to an end of this session, where we discussed about….

- Exception Handling

## Reference material

- http://www.tutorialspoint.com/python
- http://www.learnpython.org/
- http://docs.python.org/2/tutorial/

# Questions

# Key contacts

**Sakshi Jamgaonkar**

sakshi_jamgaonkar@persistent.com

**Asif Immanad**

asif_immanad@persistent.co.in



Persistent

Persistent

# Thank you!