Exception Handling

Error and its types

- Error in a software are called bugs and process of removing error is called debugging.
- We can classify errors in three types:
  - Compile time errors: These are detected by python compiler. Syntactical errors found in code, e.g. forgetting a colon in if or while statement
  - Run time errors: These are not detected by python compiler. These are detected by PVM. When PVM can’t execute the byte code, e.g. we are referring a list element by an index which is greater than the list length.
  - Logical errors: Logical error can’t detected neither by python compiler nor by PVM. It occurs when wrong formula/algorithm is used for calculation.

The try block lets you test a block of code for errors.

The except block lets you handle the error.

The else block lets you execute code when there is no error.

The finally block lets you execute code, regardless of the result of the try- and except blocks.

- When an error occurs, or exception as we call it, Python will normally stop and generate an error message.

- These exceptions can be handled using the try statement

```
try:
  print(x)
except:
  print("An exception occurred“)
```

- Since the try block raises an error, the except block will be executed.
- Without the try block, the program will crash and raise an error:

Many Exceptions

You can define as many exception blocks as you want, e.g. if you want to execute a special block of code for a special kind of error:

Example

Print one message if the try block raises a NameError and another for other errors:

```
try:
  print(x)
except NameError:
  print("Variable x is not defined")
except:
  print("Something else went wrong")
```

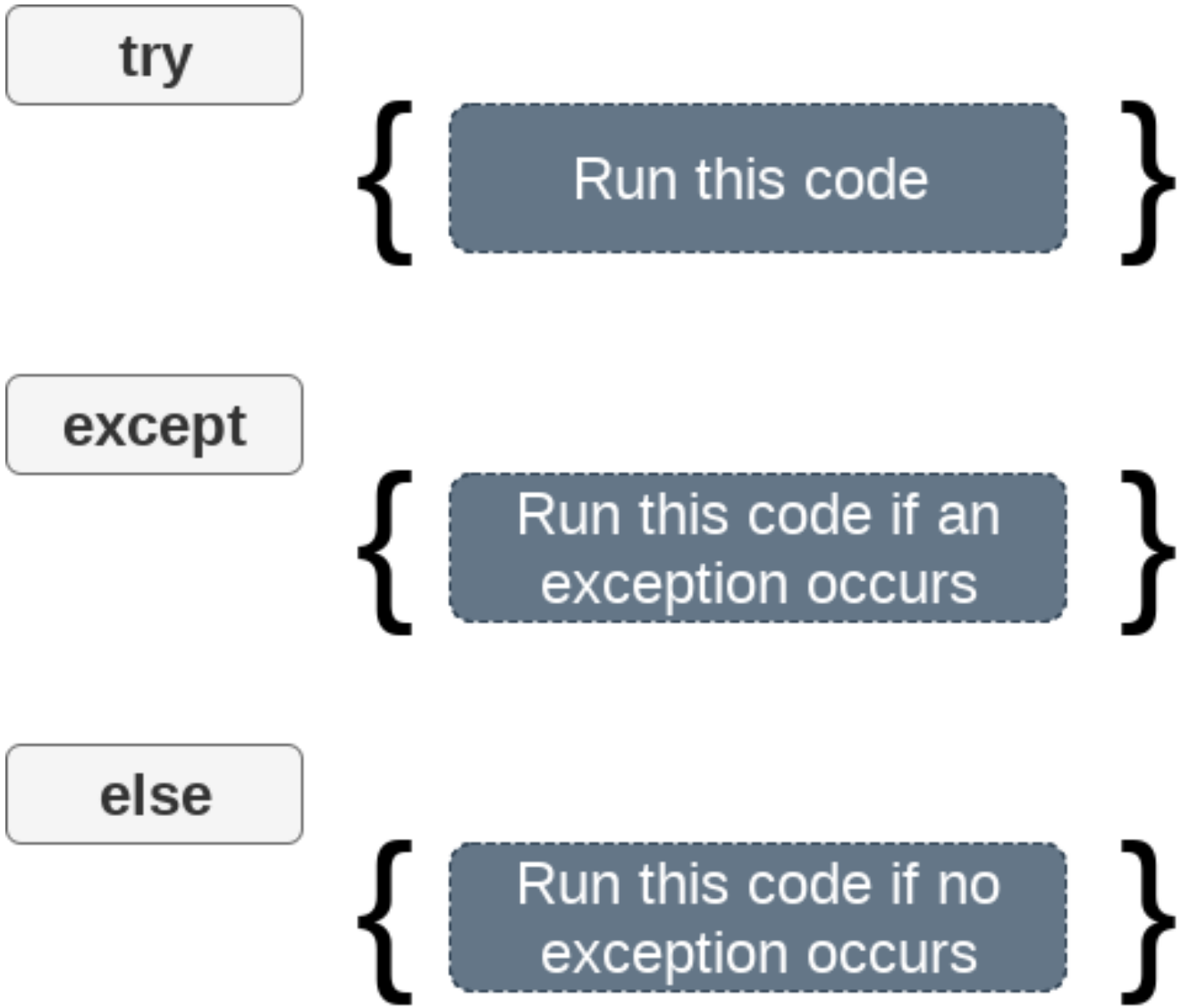
Else

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

Example

In this example, the try block does not generate any error:

```
try:
  print("Hello")
except:
  print("Something went wrong")
else:
  print("Nothing went wrong")
```


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

Finally

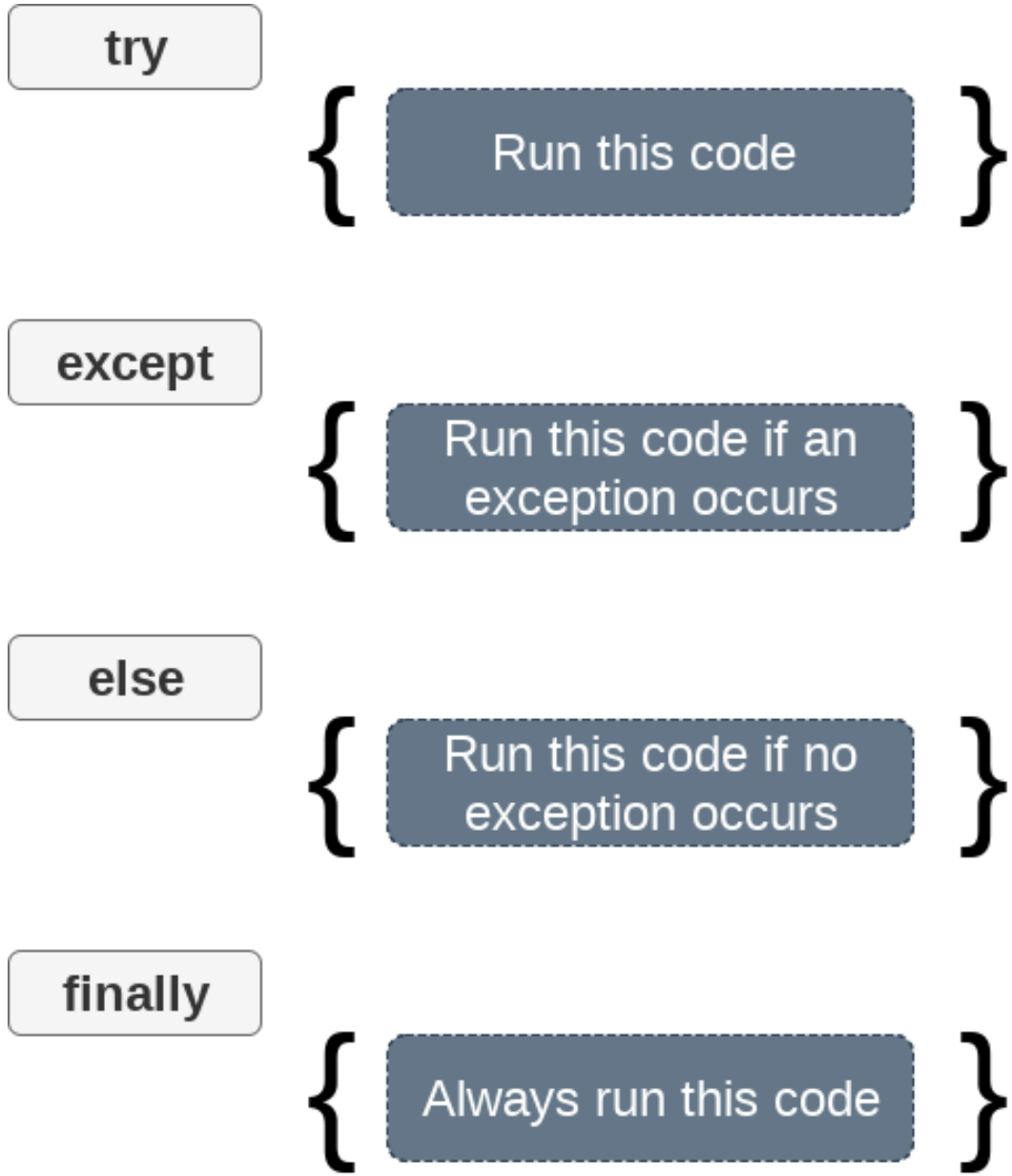
The finally block, if specified, will be executed regardless if the try block raises an error or not.

Example

```
try:
  print(x)
except:
  print("Something went wrong")
finally:
  print("The 'try except' is finished")
```

Python program to handle exception

```
try:
  f = open("demofile.txt",'w')
except:
    print("hhSomething went wrong when writing to the file")
finally:
    f.close()
    print("h")
```


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
}

Raise an exception

To throw (or raise) an exception, use the raise keyword.

Example
Raise an error and stop the program if x is lower than 0:

```
x = -1

if x < 0:
  raise Exception("Sorry, no numbers below zero")
```

You can define what kind of error to raise, and the text to print to the user.

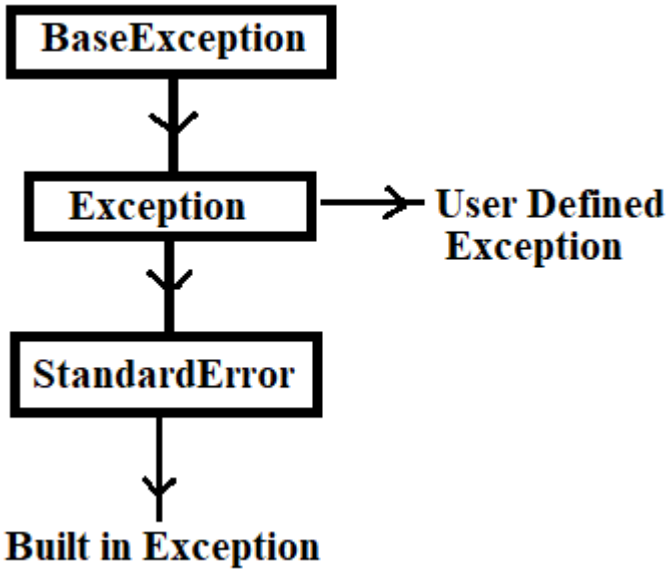
Example

Raise a TypeError if x is not an integer:

```
x = "hello"

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

Exception

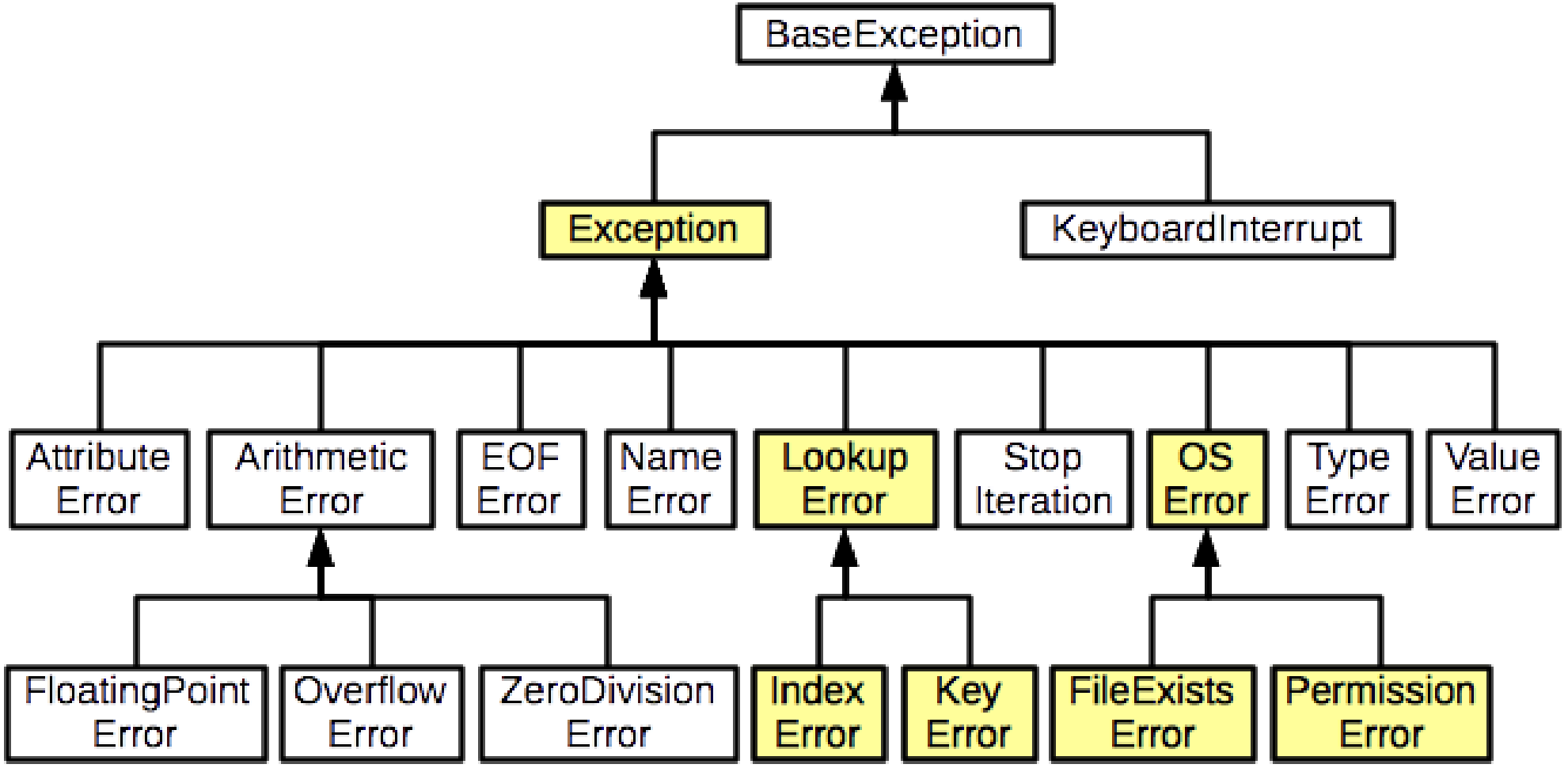


- Exception is a runtime error which can be handled by programmer.

  All exceptions are represented as classes in Python.

  - The exceptions which are already available in Python are called built-in exceptions. Base class for all built-in exceptions is **StandardError** class and it has been derived from **Exception** class. Exception class is derived from **BaseException** class.
  - An exception created by programmer is called user defined exception. All the user defined exception classes are derived from Exception class.

Exception Hierarchy



Exception Description

Exception

Base class for all exceptions

StopIteration

Raised when the next() method of an iterator does not point to any object.

ArithmeticError

Base class for all errors that occur for numeric calculation.

Exception Description

AttributeError

Raised in case of failure of attribute reference or assignment.

EOFError

Raised when there is no input from input() function and the end of file is reached.

NameError

Raised when an identifier is not found in the local or global namespace.

LookupError

Base class for all lookup errors.

IndexError

Raised when an index is not found in a sequence.

Exception Description

KeyError

Raised when the specified key is not found in the dictionary.

TypeError

Raised when an operation or function is attempted that is invalid for the specified data type.

ValueError

Raised when the built-in function for a data type has the valid type of arguments, but the arguments have invalid values specified.

OverflowError

Raised when a calculation exceeds maximum limit for a numeric type.

Exception Description

FloatingPointError

Raised when a floating point calculation fails.

ZeroDivisionError

Raised when division or modulo by zero takes place for all numeric types.

OSError

Raised for operating system-related errors.

Key points about Exception Handling

- A single try block can be followed by multiple except blocks.
- We can’t write except block(s) without a try block.
- We can write a try block without any except blocks.
- Else blocks and finally blocks are not compulsory.
- When there is no exception, else block is executed after try block.
- Finally block is always executed (whether there is exception or not)

Assert Statement

- It is used to ensure that a given condition is true. If it is not true, it raises AssertionError. Syntax is as follows:

  assert condition, message

- If the condition is false then the exception by the name AssertionError is raised along with the message written in the assert statement.

3.31-Python program to show assert statement

```
try:
    x=int(input('Enter a number between 5 and 10: '))
    assert x>=5 and x<=10, "your input is not correct"
    print('the number entered is ',x)
except AssertionError as obj:
    print(obj)
else:
    print("No exception")
finally:
    print("finally executed")
```

```
Enter a number between 5 and 10: 6
the number entered is  6
No exception
finally executed
>>>
```

```
Enter a number between 5 and 10: 4
your input is not correct
finally executed
>>>
```