Exception Handling

What is **Exceptions**?

- Official explanation: An exception is an event, which occurs during the execution of a program, that disrupts the normal flow of the program's instructions.
- So ... it is an unexpected event, like an error. However, not all exceptions are errors.
- When an error occurs an exception is raised.
- When an exception occurs, python stops whatever it is doing and goes to the last seen exception handler.
- You can raise exceptions yourself in the code.
- You can create your own exceptions.
- When an exception is handled the program can continue.

Finally - Exceptions

- Exceptions – errors thrown during execution
- If an exception is not handled, execution stops with an error message.
- Some of the more important exceptions are:

Exceptions	Meaning
AttributeError	Object does not contain the specified instance variable or method
ImportError	The import statement fails to find a specified module or name in that module
IndexError	A sequence (list, string, tuple) index is out of range
KeyError	Specified key does not appear in a dictionary
NameError	Specified local or global name does not exist
TypeError	Operation or function applied to an inappropriate type
ValueError	Operation or function applied to correct type but inappropriate value
ZeroDivisionError	Second operand of division or modulus operation is zero

Handling an exception with try/except statement

- If you have some *suspicious* code that may raise an exception, you can defend your program by placing the suspicious code in a **try:** block. After the try: block, include an **except:** statement, followed by a block of code which handles the problem as elegantly as possible.

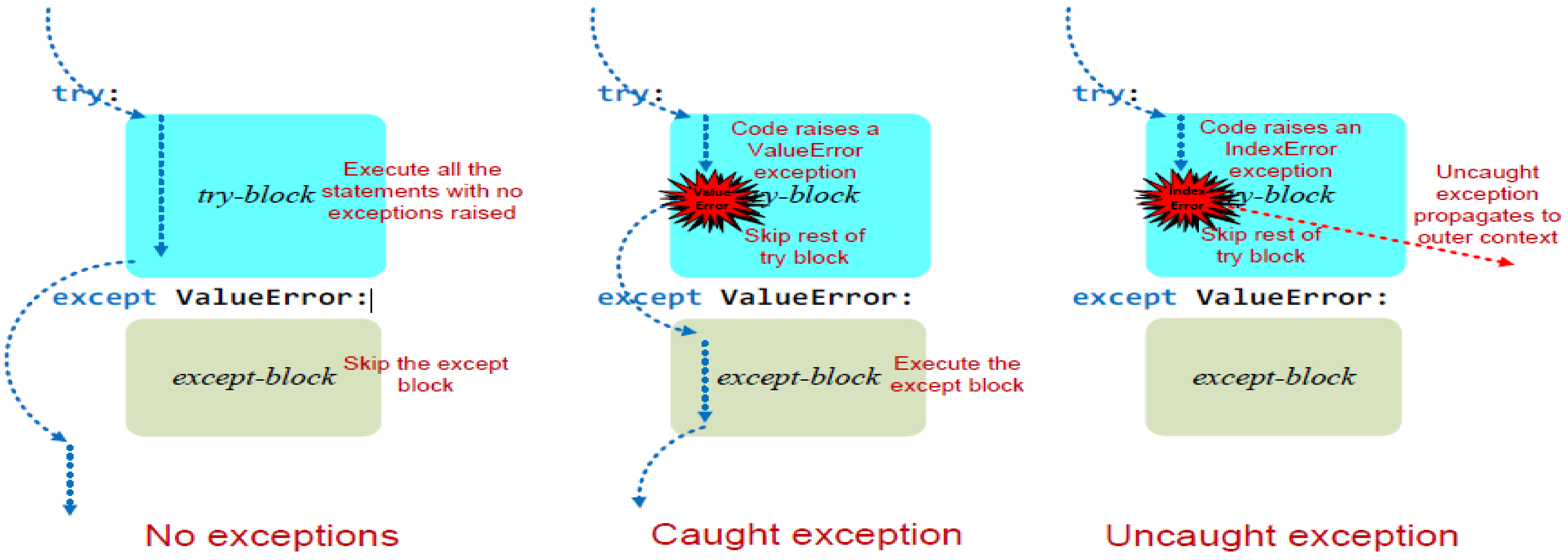
Syntax:

```
try:
   You do your operations here;
   ......................
except ExceptionName:
   If there is ExceptionName, then execute this block.
```

Example:

```
try:
   val = int(input("Please enter a small positive integer: "))
   print('You entered', val)
except ValueError:
   print('Input not accepted')
```

The possible program execution flows through a try/except statement



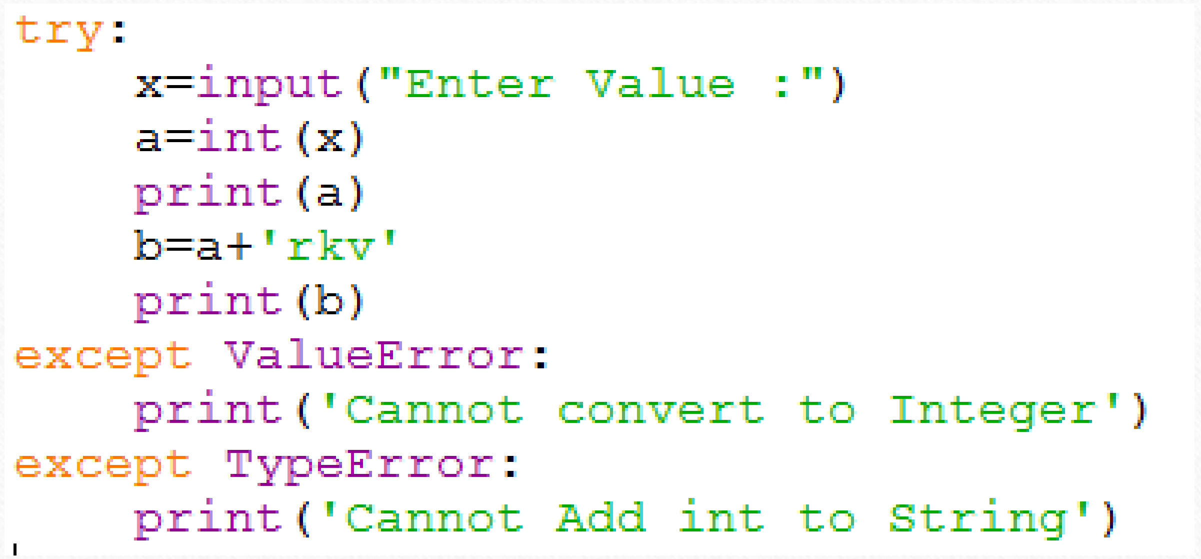
Handling Multiple Exceptions

- A **try** statement can have multiple **except** blocks. Each except block must catch a different type of exception object.

Syntax:

```
try:
   You do your operations here;
   ......................
except Exception I:
   If there is ExceptionI,
   then execute this block.
except Exception II:
   If there is ExceptionII,
   then execute this block.
   ......................
```

Example:



```
try:
    x=input("Enter Value :")
    a=int(x)
    print(a)
    b=a+'rkv'
    print(b)
except ValueError:
    print('Cannot convert to Integer')
except TypeError:
    print('Cannot Add int to String')
```

The **except** clause with multiple exceptions

- You can also use the same *except* statement to handle multiple exceptions as follows:

Syntax:

```
try:
   You do your operations here;
   ......................
except(Exception1[, Exception2[,...ExceptionN]]]):
   If there is any exception from the given exception list, then execute this block
   .......................
```

Example:

```
try:
   x=input("Enter Value:")
   a=int(x)
   print a
   b=a+'rkv'
   print b
except (ValueError, TypeError):
   print 'Problem in the Code'
```

Catching All Exceptions

- To catch all exceptions, write an exception handler for **Exception**,

Example -1:

```
try:
   a=int('x')
except Exception :
   print 'Problem in the Code'
```

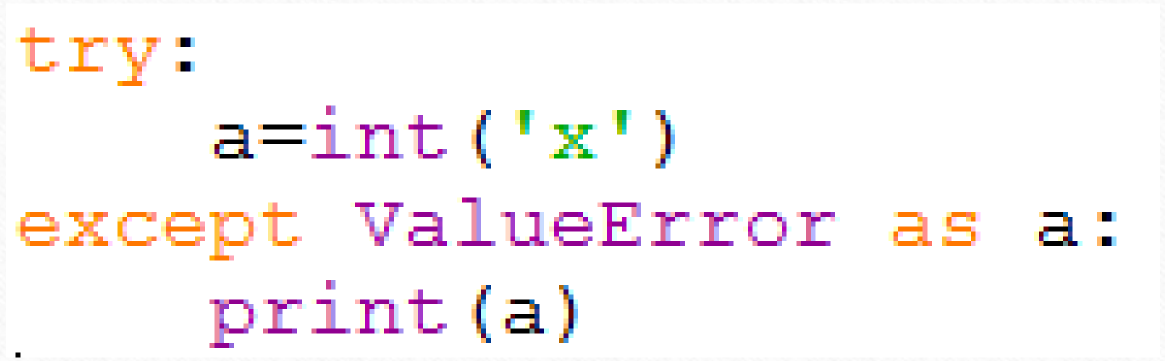
Example -2:

```
try:
   a=int('x')
except:
   print 'Problem in the Code'
```

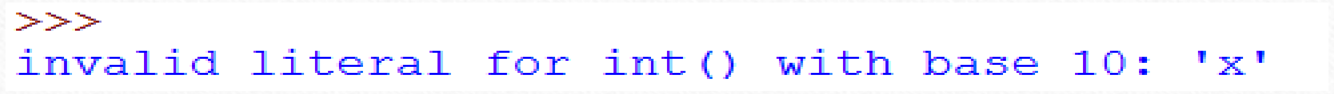
Some system exiting exceptions like KeyboardInterrupt or SystemExit would be caught by except: and not by except Exception: as they are based on Base Exception

Catching Exception Objects

- The **except** blocks “catch” exception objects. We can inspect the object that an except block catches if we specify the object’s name with the **as** keyword.



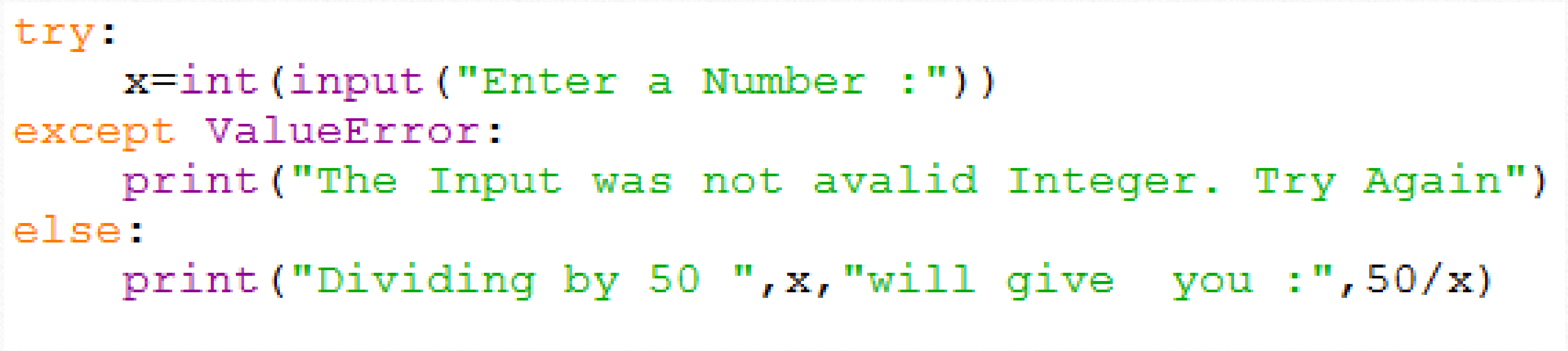
```
try:
    a=int('x')
except ValueError as a:
    print(a)
```



```
>>>
invalid literal for int() with base 10: 'x'
```

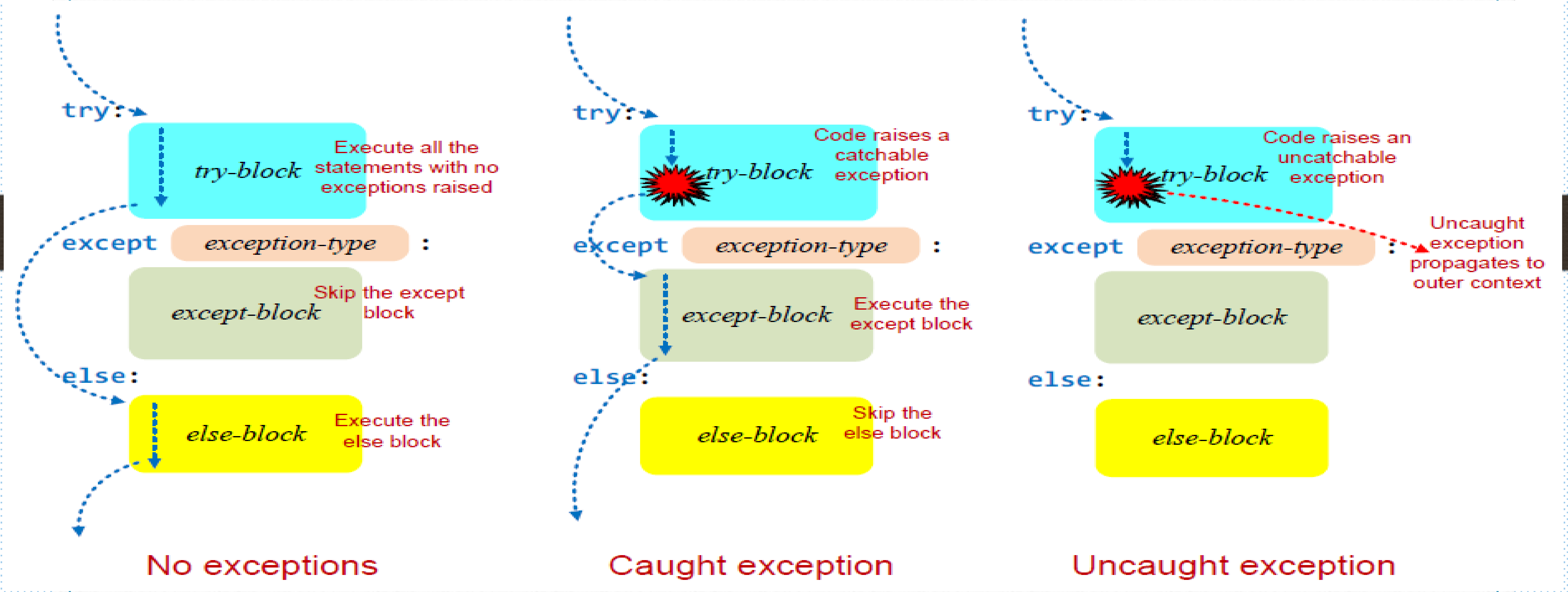
The try Statement’s Optional else Block

- The try ... except statement has an optional else clause. An else block has to be positioned after all the except clauses. An else clause will be executed if the try clause doesn't raise an exception.



```
try:
    x=int(input("Enter a Number :"))
except ValueError:
    print("The Input was not avalid Integer. Try Again")
else:
    print("Dividing by 50 ",x,"will give  you :",50/x)
```

The possible program execution flows through a try/else statement

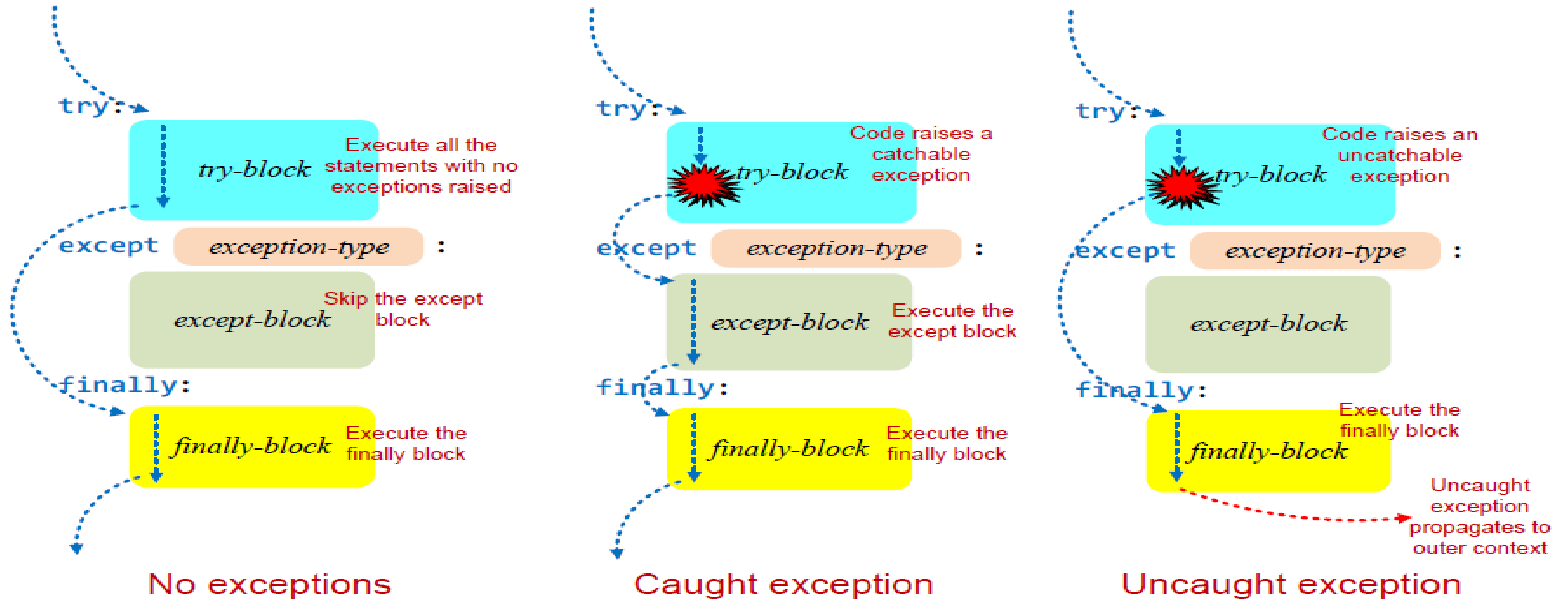


finally block

- A **try** statement may include an optional **finally** block. Code within a **finally** block always executes whether the **try** block raises an exception or not. A **finally** block usually contains "clean-up code" that must execute due to activity initiated in the **try** block.

```
try:
    x = int(input("Please enter a number: "))
except ValueError:
    print("The input was not a valid integer. Please try again...")
else:
    print("Dividing 50 by", x,"will give you :", 50/x)
finally:
    print("Already did everything necessary.")
```

The possible program execution flows through a try/except/finally statement



Raising Exceptions

- The **raise** keyword raises an exception.

Syntax:

raise [Exception [, args [, traceback]]]

(Here, Exception is the type of exception (for example, NameError) and argument is a value for the exception argument. The argument is optional; if not supplied, the exception argument is None.)

Example:

```
try:
    a = int(input("Enter a positive integer: "))
    if a <= 0:
        raise KeyError("That is not a positive sfsdfsd number!")
except KeyError as ve:
     print(ve)
```

Exception structure

The try-except structure of error handling

```
try:
    normal statements which may fail
except exception-type:
    error handling statements for this specific exception
except (exception-type1, exception-type2):
    error handling statements for these specific exceptions
except exception-type as errname:
    error handling statements for this specific exception getting instance via errname
except:
    general error handling statements catching all exception types
else:
     statements executed if no exception
finally:
     clean-up statements always executed
```

Example Programs in Exceptions

- What is exception? Describe exceptions with examples.
- How to handle exception? Explain the execution flows through a try/except statement when no exception, caught exception and uncaught exception with example
- Explain try statement …. else block with example? Also explain the execution flows through a try/else statement when no exception, caught exception and uncaught exception with example
- Explain try statement …. finally block with example? Also explain the execution flows through a try/finally statement when no exception, caught exception and uncaught exception with example
- What is raise exception? What is the use with it?
- Describe the try-except structure of error handling?