EXCEPTION HANDLING — Exam Cheat Sheet

Class XII Computer Science (083) | try, except, else, finally, raise, assert, output questions

1. What and why

- An **exception** is an error that happens while the program is RUNNING, e.g. dividing by zero or opening a file that does not exist.
- If an exception is not handled, Python stops the program and prints an error message (a traceback). Lines after the error never run.
- **Exception handling** means catching the error with try and except, so the program shows a proper message and continues.

	Syntax error	Exception (runtime error)
When found	before the program runs	while the program is running
Cause	rules of Python broken	a statement cannot be carried out
Example	print("Hi" or if x > 5	10 / 0 or int("abc")
Can try-except catch it?	No — fix the code	Yes

CBSE syllabus: Introduction, handling exceptions using try-except-finally blocks. NCERT also covers else, raise and assert.

2. Built-in exceptions you must know

Exception	Raised when	Example
`ZeroDivisionError`	dividing or taking % by zero	`10 / 0`
`ValueError`	right type but wrong value	`int("abc")`
`TypeError`	operation on the wrong type	`"5" + 5`
`NameError`	variable not defined	`print(total)`
`IndexError`	list / string index out of range	`L = [1, 2]; L[5]`
`KeyError`	dictionary key not found	`D = {'a': 1}; D['b']`
`FileNotFoundError`	opening a missing file for reading	`open("x.txt")`
`IOError`	input / output operation fails	`file cannot be read or written`
`EOFError`	end of file reached while reading	`pickle.load(f) at the end`
`ImportError`	module cannot be found	`import maths`
`OverflowError`	result too large to store	`2.0 ** 10000`
`IndentationError`	wrong indentation (a syntax error)	`line not indented after if`

3. The four blocks

```
try:
    # statements that may cause an error
except ExceptionName:
    # runs ONLY if that error occurs in try
else:
    # runs ONLY if NO error occurs in try
finally:
    # runs ALWAYS – error or no error
```

Block	Compulsory?	When it runs
`try`	yes	always runs first
`except`	at least one except OR a finally	only when a matching error occurs in try
`else`	optional, after all except blocks	only when try finishes with no error
`finally`	optional, always last	always — used for closing files and clean-up

4. Template A — one except block

```
try:
    a = int(input("Enter a number: "))
    b = int(input("Enter another number: "))
    print("Quotient =", a / b)
except ZeroDivisionError:
    print("Cannot divide by zero")
print("Program continues")
```

5. Template B — many except blocks

```
try:
    a = int(input("Enter a number: "))
    b = int(input("Enter another number: "))
    print("Quotient =", a / b)
except ZeroDivisionError:              # specific error first
    print("Cannot divide by zero")
except ValueError:
    print("Please enter numbers only")
except (TypeError, NameError):         # one block for two errors
    print("Type or name problem")
except:                                # catches anything else — keep LAST
    print("Some other error")
```

Python checks except blocks from top to bottom and runs only the FIRST one that matches. A bare except: must be the last.

6. Template C — else and finally

```
try:
    f = open("data.txt", "r")
    print(f.read())
except FileNotFoundError:
    print("File not found")
else:
    print("File read successfully")    # only when no error
    f.close()
finally:
    print("End of program")            # always
```

Showing Python's own error message

```
try:
    print(10 / 0)
except ZeroDivisionError as e:         # e holds the error message
    print("Error:", e)                 # Error: division by zero
```

7. Output questions — predict the output

Q1. Which lines print?

```
try:
    x = 10 / 0
    print("A")
except ZeroDivisionError:
    print("B")
finally:
    print("C")
```

Output:

```
B
C
```

print("A") is skipped: once an error occurs, the rest of try is skipped.

Q2. No error in try

```
try:
    n = int("25")
    print(n + 5)
except ValueError:
    print("Invalid")
else:
    print("No error")
finally:
    print("Done")
```

Output:

```
30
No error
Done
```

No error, so else runs, then finally.

Q3. Which except block runs?

```
L = [10, 20, 30]
try:
    print(L[3])
except ValueError:
    print("Value")
except IndexError:
    print("Index")
except:
    print("Other")
```

Output:

```
Index
```

Index 3 does not exist. Only the first matching block runs.

Q4. finally with return

```
def f(a, b):
    try:
        return a // b
    except ZeroDivisionError:
        return -1
    finally:
        print("End", end=" ")

print(f(7, 2))
print(f(7, 0))
```

Output:

```
End 3
End -1
```

finally runs even before return, so "End" prints first.

Q5. Error inside a loop

```
for x in [4, 0, 2]:
    try:
        print(8 // x, end=" ")
    except ZeroDivisionError:
        print("Inf", end=" ")
```

Output:

```
2 Inf 4
```

The error is handled inside the loop, so the loop carries on.

Q6. Dictionary key

```
D = {'a': 1}
try:
    print(D['b'])
except KeyError:
    print("Key missing")
else:
    print("Found")
```

Output:

```
Key missing
```

An error occurred, so else does not run.

Q7. raise with a message

```
try:
    age = -5
    if age < 0:
        raise ValueError("Age cannot be negative")
    print("Age OK")
except ValueError as e:
    print("Error:", e)
```

Output:

```
Error: Age cannot be negative
```

raise jumps straight to the except block.

Q8. Error not caught

```
try:
    print("Start")
    x = int("abc")
except ZeroDivisionError:
    print("Zero")
finally:
    print("Finally")
print("End")
```

Output:

```
Start
Finally
ValueError: invalid literal for int() with base 10: 'abc'
```

No except matches ValueError. finally still runs, then the program stops, so "End" never prints.

8. Quick reference — which blocks run?

Situation	try	except	else	finally
No error in try	full	skipped	runs	runs
Error, matching except found	stops at error	that one runs	skipped	runs
Error, no matching except	stops at error	none	skipped	runs, then program crashes

9. Marks are lost here — check every time

- Writing except before try, or else / finally before except — the order is always try → except → else → finally.
- Putting a bare except: before a specific one — it catches everything, so later blocks never run.
- Thinking the rest of try runs after an error — everything after the error line in try is skipped.
- Forgetting that finally runs ALWAYS, even when there is a return or an error that is not caught.
- Expecting else to run after an error — else runs only when try has NO error.
- Mixing up error names: int("abc") is ValueError, "5" + 5 is TypeError, undefined variable is NameError.
- Saying try-except catches syntax errors — the program must first be free of syntax errors to run at all.
- Spelling the name wrong — ZeroDivisionError, not ZeroDivideError; FileNotFoundError, not FileNotFound.
- Writing raise ValueError, "msg" — the message goes inside brackets: raise ValueError("msg").