PYTHON ERRORS AND EXCEPTIONS HANDLING

INTRODUCTION

Error and exception handling is a crucial part of programming in Python.

It helps ensure that your code behaves predictably even when unexpected situations arise.

Types of Errors in Python

- Syntax Errors
- Semantic Errors
- Run Time Errors
- Logical Errors

SYNTAX ERRORS

Syntax errors refer to formal rules governing the construction of valid statements in a language.

Syntax errors occur when rules of a programming language are misused i.e., when a grammatical rule of the language is violated.

SEMANTIC ERRORS

Semantics error occur when statements are not meaningful

Semantics refers to the set of rules which give the meaning of the statement.

Example

- X * Y = Z
- will result in semantical error as an expression can not come on the left side of an assignment statement

RUN TIME ERRORS

A Run time error is that occurs during execution of the program. It is caused because of some illegal operation taking place.

For example

- If a program is trying to open a file which does not exists or it could not be opened(meaning file is corrupted), results into an execution error.
- An expression is trying to divide a number by zero are RUN TIME ERRORS.

LOGICAL ERRORS

A Logical Error is that error which is causes a program to produce incorrect or undesired output.

for instance,

```
        ctr=1;
        while(ctr<10):
                print(n *ctr)
```

EXCEPTIONS

INTRODUCTION

Occur when syntactically correct code results in an error.

Example: dividing by zero, accessing an index that doesn't exist.

- a = 5 / 0

COMMON BUILT-IN EXCEPTIONS

ZeroDivisionError: When division or modulo by zero occurs.

TypeError: Invalid operation on types.

ValueError: Function gets an argument of the correct type but inappropriate value.

IndexError: List index out of range.

KeyError: Dictionary key is not found.

IOError: An I/O operation (like file read/write) fails.

AttributeError: Object doesn’t have the requested attribute.

EXCEPTION HANDLING

INTRODUCTION

Python provides a way to handle exceptions using *try, except, else, finally* blocks.

```
try:
    # Code that may raise an exception
except SomeException:
    # Handle the exception
```

Example:

```
try:
    x = 10 / 0
except ZeroDivisionError:
    print("You can't divide by zero!")
```

HANDLING MULTIPLE EXCEPTIONS

You can catch multiple exceptions with a single except or handle them individually.

```
try:
    x = int("abc")
except ValueError:
    print("Invalid value!")
except ZeroDivisionError:
    print("You can't divide by zero!")
```

USING ELSE AND FINALLY

else: Executed if no exception is raised.

finally: Always executed, regardless of whether an exception was raised.

```
try:
    result = 10 / 2
except ZeroDivisionError:
    print("Can't divide by zero!")
else:
    print("Division successful:", result)
finally:
    print("Execution finished.")
```

RAISING EXCEPTIONS

You can raise exceptions in your code using the raise keyword.

```
x = -5
if x < 0:
    raise ValueError("Negative numbers are not allowed")
```

CREATING CUSTOM EXCEPTIONS

You can create your own exception classes by inheriting from the base Exception class.

```
class CustomError(Exception):
    pass

try:
    raise CustomError("This is a custom error!")
except CustomError as e:
    print(e)
```