3.1 ERRORS AND EXCEPTIONS

3.2.1 Errors

Errors or mirrors in a program are often referred to as bugs. Error is mistakes in the program, which are created by fault of the programmer.

Debugging:

Debugging is the process of finding and eliminating errors.

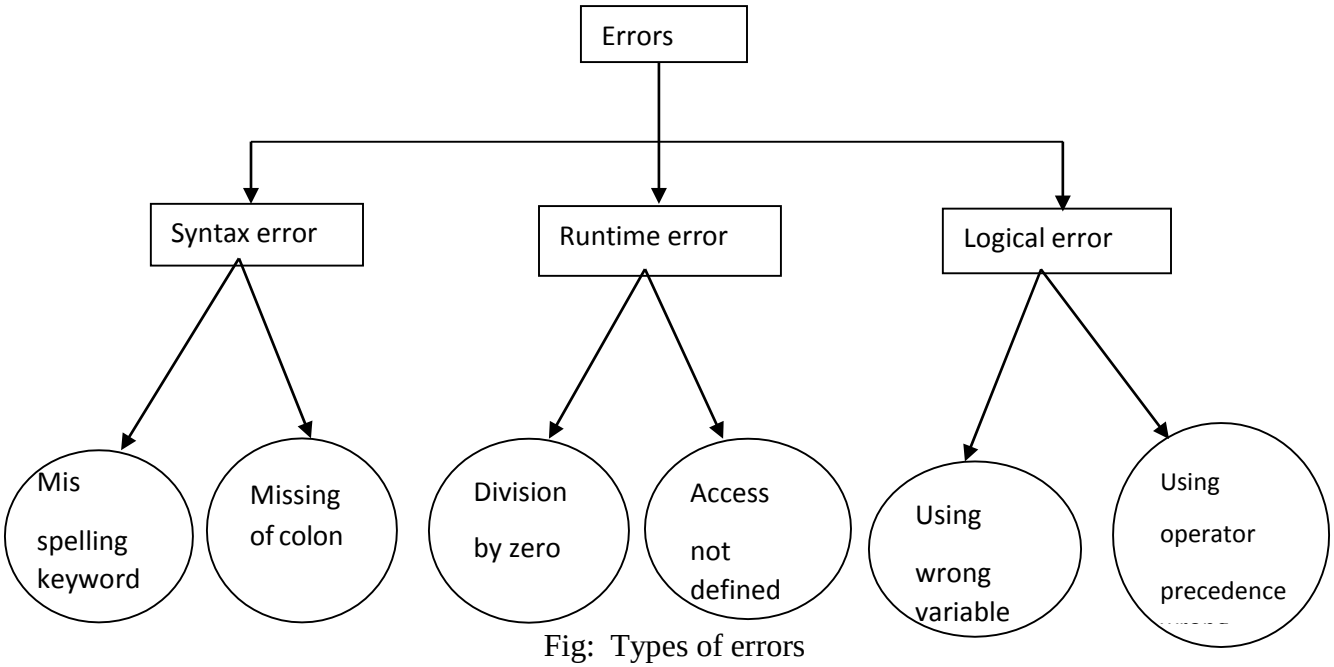

Fig: Types of errors

Types of errors:

i) Syntax errors
ii) Runtime errors
iii) Logical error

i) Syntax errors

Syntax error occurs when the program is not following the proper structure (syntax). If syntax errors occur then python display error message and exit without continuing execution process.

Some of syntax errors are:

- Misspelling keyword
- Incorrect indentation
- Missing of colon, comma
- Leaving a keyword
- Putting a keyboard in wrong place

Here are some examples of syntax errors in Python:

```
a=5
b=20
if a<b
print 'a is greater'
```

Error Message:

```
File "main.py", line 3
if a<b
```

ii) Runtime errors

Runtime errors occur during execution of program. It is also known as dynamic errors.

Some of runtime errors are:

- Division by Zero
- Access not defined identifier
- Access not defined file

Example:

```
>>>b= [1,2,3,4]
>>>a
Runtime Error: 'a' is not defined
>>>a=5
>>>b=10
>>>a/b
Runtime Error: 'Division by Zero'
```

iii) Logical errors

Logical error occurs due to mistake in program's logic. Here program runs without any error messages, but produces an incorrect result. These errors are difficult to fix. Here are some logical errors are:

- Using the wrong variable name
- Indenting a block to the wrong level
- Using integer division instead of floating-point division
- Using wrong operator precedence
- Making a mistake in a Boolean expression
- Off-by-one and other numerical errors

Example:Addition of two numbers

```
>>>x=10
>>>y=20
>>>z=x-y      #Logic error
>>>print z
  -10
```

Here, parentheses are missing. So we get wrong result.

3.2.2 Exceptions

An exception is an error that occurs during execution of a program. It is also called as runtime error**.** Some examples of Python runtime errors:

- division by zero
- performing an operation on incompatible types
- using an identifier which has not been defined
- accessing a list element, dictionary value or object attribute which doesn't exist
- trying to access a file which doesn't exit

An example for run time error is as follows:

```
print (10/0)
Error Message:
Traceback (most recent call last): File "main.py", line 1, in <module>
print (10/0)
```

ZeroDivisionError: integer division or modulo by zero

Exceptions come in different types, and the type is printed as part of the message: the typein the example is ZeroDivisionError which occurs due to division by 0. The string printed as the exception type is the name of the built-in exception that occurred. Exception refers to unexpectedcondition in a program. The unusual conditions could be faults, causing an error which in turn causes the program to fail. The error handling mechanism is referred to as exception handling. Many programming languages like C++, PHP, Java, Python, and many others have built-in supportfor exception handling.

Python has many built-in exceptions which forces your program to output an error whensomething in it goes wrong. When these exceptions occur, it stops the current process and passes the control to corresponding exception handler. If not handled, our program will crash.

Some of the standard exceptions available in Python are listed below.

Exception Name	Description
Exception	Base class for all exceptions
ArithmeticError	Base class for all errors that occur for numeric calculation.
OverflowError	Raised when a calculation exceeds maximum limit for a numeric type.
FloatingPointError	Raised when a floating point calculation fails.
ZeroDivisionError	Raised when division or modulo by zero takes place for all numeric types.
AssertionError	Raised in case of failure of the Assert statement
EOFError	Raised when there is no input from either the raw_input() or input() function and the end of file isreached.
ImportError	Raised when an import statement fails.
IndexError	Raised when an index is not found in a sequence

KeyError	Raised when the specified key is not found in the dictionary.
NameError	Raised when an identifier is not found in the local or global namespace
IOError	Raised when an input/ output operation fails, such as the print statement or the open() function when trying to open a file that does not exist.
SyntaxError	Raised when there is an error in Python syntax
SystemExit	Raised when Python interpreter is quit by using the sys.exit() function. If not handled in the code, causes the interpreter to exit
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.
RuntimeError	Raised when a generated error does not fall into any category