Python Errors:
In python there are three types of errors
1. Syntax Errors
2. Logical Errors or Semantic Errors
3. RunTime Errors

1 Syntax Errors
These are the most basic type of errors. Syntax errors are almost always fatal. In IDLE, it will highlight where the syntax error is. Most syntax errors are typos, incorrect indentation/incorrect arguments.

Example:
```
print(Hello, World!)
print("Hello')
print('Hello")
class="Naresh i Technologies"
Name="KSRaju
```

Here are some ways to avoid the most common Syntax Errors:
1 Never use a Python keyword for a variable name
2 Any strings in the code should have matching quotation marks
3 Always check unclosed opening operators - (,{.(
4 Always check the indentation
5 Don't give your module the same name as one of the standard Python modules.

2 Logical Errors:
Your program might run without crashing (no syntax or run-time errors), but still do the wrong thing.

Example:
```
x = 3; y = 4
avg = x + y / 2
print(avg)
```

but the program prints 5.0 instead!
x+ y / 2, this has the same mathematical meaning as x + (y / 2) = 3 + (4 / 2) = 3 + 2 = 5. To fix the problem, average = (x + y) / 2, which makes clear. Logic errors can be difficult to spot, especially in a longer program, but as you get better at writing code you will also get better at avoiding logic errors.

3. RunTime Errors:
Python is able to understand what the program says, but runs into problems when actually performing the instructions. Like spelling mistakes or invalid Methods & Properties etc...!!

Example:
```
x=500
printf(x)

CallMe="KSRaju"
print(callme)
```

Common RunTime Errors in PYTHON:

In PYTHON The following are common RunTimeErrors:
1 NameError 2 TypeError
3 KeyError 4 AttributeError
5 IndexError

Abnormal termination:
The concept of terminating the program or script in the middle of its execution without executing last statement(s) of the program is known as abnormal termination.

Example:
```
a=input("Enter any Number")
b=input("Enter any Number")
c=a/b
print("The Value is: " ,c)
```

NOTE:
TypeError: unsupported operand type(s) for /: 'str' and 'str'

NOTE:
Abnormal termiation is the undesirable situation in any Programming language.

Types of Exceptions:
In Python there are 2 types of exceptions are possible.
1.Predefined Exceptions 2.User Definded Exceptions

Predefined Exceptions: Also known as in-built exceptions
Python Exception Handling
Python provides two important features to handle any unexpected error in your Python programs:
1 Exception Handling 2 Assertions

What is an Exception?
An exception is an error that happens during execution of a program. When that error occurs, Python generate an exception that can be handled, which avoids your program to crash.

Common Exception Errors
```
1 except IOError:
    print('An error occurred trying to read the file.')

2 except ValueError:
    print('Non-numeric data found in the file.')

3 except ImportError:
    print("NO module found")

4 except EOFError:
    print('End Of File No Data')

5 except KeyboardInterrupt:
    print('You cancelled the operation.')

6 except:
    print('An error occurred.')
```

Example:

```
number=int(input("Enter Any Number: "))
print("You Are Entered: ", number)
```

Note:
The above script get executed perfectly as long as the user enters a number, If not It terminates, then we should develop exception logic.

Syntax:

```
try:
    Some Statements Here
except:
    Exception Handling
```

Example:

```
try:
  x=int(input("Enter a value: "))
  print("Try Block:Enter Number is: ",x)
except ValueError:
  print("ExceptBlock:Invalid Input Enter Only Numbers");
```

Example:

```
try:
    a=int(input("Enter any Number: "))
    b=int(input("Enter any Number: "))
    c=a/b
    print("The Value is: " ,c)
except ValueError:
    print("SooryInvalidInput")
```

Example:

```
try:
    a=int(input("Enter any Number: "))
    b=int(input("Enter any Number: "))
    c=a/b
    print("The Value is: " ,c)
except ValueError:
    print("SooryInvalidInput")
except ZeroDivisionError:
    print("b must be valid Number")
```

Example:

```
try:
  x=int(input("Enter Required Number: "))
  y=int(input("Enter Required Number: "))
  print("The Result is: ",x/y)
except Exception as arg:
    print("You can't divide by zero.")
    print('Error: ', arg)
```

try with multiple except blocks:
Example:

```
try:
    p=int(input("Enter Any Number: "))
    q=int(input("Enter Any Number: "))
    r=p/q
    print("The Value is: ",int(r))
```

```
    print("TryBlockIsExecuted:")
except ValueError:
    print("ExceptBlock")
    print("Sorry User Numbers Only Valid")
except Exception as arg:
    print("You can't divide by zero.")
    print('Error: ', arg)
```

Syntax-II

```
try:
   You do your operations here;
   ......................
except Exception-I:
   If there is ExceptionI, then execute this block.
   ......................
else:
   If there is no exception then execute this block.
```

Example:

```
try:
    fi=open("Hai1.txt",'r')
    print(fi.read(6))
except IOError:
    print("File Not Existed Please Create")
else:
    print("ContentReadSuccessfully")
    fi.close()
```

NOTE: If file Not Existed It returns except IOError

Example:

```
try:
    x=int(input("Enter Any Input: "))
    y=int(input("Enter Any Input: "))
    z=x/y
    print(z)
except ValueError:
    print("SorryAlphabetsUnableToCompute")
    print("Special Chars Also Not Accepted")
except Exception as arg:
    print("Error: ",arg)
else:
    print("Welcome to Else Block")
    print("Else Block Successfully Executed")
    print("Welcome to More Computations")
    print(x+y)
    print(x-y)
    print(x*y)
```

Default except block:

The except Clause with No Exceptions:

You can also use the except statement with no exceptions:

Syntax-III

```
try:
   You do your operations here;
   ......................................
```

```
   ..........................................
except:
   If there is any exception, then execute this block.
   ..........................................
   ..........................................
else:
   If there is no exception then execute this block.
```

Example:

```
try:
  x=int(input("Enter Required Number: "))
  y=int(input("Enter Required Number: "))
  print("The Result is: ",int(x/y))
except:
    print("Arithmetic Exception Raised.")
else:
    print("SuccessfullyDone")
```

Syntax:-IV
The try-finally Clause
You can use a finally: block along with a try: block. The finally block is a place to put any code that must execute, whether the try-block raised an exception or not.

```
try:
   You do your operations here;
  ..........................................
except:
Exceptional Statements
  ..........................................
finally:
   This would always be executed.
  ..........................................
```

NOTE: You cannot use else clause, along with a finally clause.

Example:

```
try:
    print("try block")
except:
    print("except block")
finally:
    print("finally block")
```

SyntaxV:
Nested try-except-finally blocks:
We can take try-except-finally blocks inside try or except or finally blocks.i.e nesting of try- except-finally is possible.

Syntax:

```
try:
----------
----------
     try:
----------
----------
    except:
```

```
  ----------
  ----------
   finally:
  -----------
except:
  -----------
  -----------
finally:
-------------
-------------
```

EXAMPLE:

```
try:
    print("OuterTryBlock")
    try:
        print("InnerTryBlock")
    except ZeroDivisionError:
        print("InnerExceptBlock")
    finally:
        print("InnerFinallyBlock")
except:
    print("OuterExceptBlock")
finally:
    print("OuterFinallyBlock")
```