1. What is Python ValueError?

- Python ValueError is raised when a function receives an argument of the correct type but an inappropriate value.

2. ValueError Example:

- You will get ValueError with mathematical operations, such as square root of a negative number.

Example:

```
import math
m = math.sqrt(-10)
print(m)
```

Output:

ValueError: math domain error

3. Handling ValueError Exception

- Here is a simple example to handle ValueError exception using try-except block.

Example:

```
import math
x = None
try:
   x = int(input('Please enter a positive number:\n'))
   print(f'Square Root of {x} is {math.sqrt(x)}')
except ValueError as e:
   print(e)
   if x:
      print(f'You entered {x}, which is not a positive number.')
```

---->> Here is the output of the program with different types of input

Testing 1:

```
Please enter a positive number:
16
Square Root of 16 is 4.0
```

Testing 2:

```
Please enter a positive number:
-16
math domain error
You entered -16, which is not a positive number.
```

Testing 3:

```
Please enter a positive number:
abc
invalid literal for int() with base 10: 'abc'
```

Note:

Our program can raise ValueError in int() and math.sqrt() functions. So, we can create a nested try-except block to handle both of them. Here is the updated snippet to take care of all the ValueError scenarios.

Example:

```
import  math
try:
```

```
    x = int(input('Please enter a positive number:\n'))
    try:
        print(f'Square Root of {x} is {math.sqrt(x)}')
    except ValueError as ve:
        print(f'You entered {x}, which is not a positive number.')
except ValueError as ve:
    print(ve)
```

Testing 1:

```
Please enter a positive number:
abc
invalid literal for int() with base 10: 'abc'
```

Testing 2:

```
Please enter a positive number:
-16
You entered -16, which is not a positive number.
```

NameError:

This exception is raised when a variable isn't located in either local or global namespace.

Example:

>>> x

NameError: name 'x' is not defined

KeyError:

When the given key is not found in the dictionary object or set object , this exception is raised.

Example:

>>> dict_data = {'name':'Srinivas', 'company':'TCS'}

>>> dict_data['salary']

KeyError: **'salary'**

IndexError:

This exception is raised when the index attempted to be accessed is not found.

Example:

>>> lst=[10,20,30]

>>> lst[5]

IndexError: list index out of range

ZeroDivisionError:

For all numeric data types, its value is raised whenever a number is attempted to be divided by zero.

Example:

>>> 10/0

ZeroDivisionError: division by zero

SyntaxError:

This exception is raised whenever a syntax error occurs in our program.

Example:

>>> a 10

SyntaxError: invalid syntax

>>> print('Hiii')

SyntaxError: invalid syntax

TypeError:

This exception is raised whenever a data type-incompatible action or function is tried to be executed.

Example:

>>> 10 + 'hi'

TypeError: unsupported operand type(s) for +: 'int' and 'str'

RuntimeErrors:

This exception is raised when an error that occurred during the program's execution cannot be classified.

Q. Does exception catch RuntimeError in Python?

Whenever a RuntimeError occurs, Python creates an exception object. It then creates and prints a traceback of that error with details on why the error happened.

KeyboardInterrupt:

If the user interrupts the execution of a program, generally by hitting **Ctrl+C** , this exception is raised.

Example:

```
def message():
   print('Hello')
   message()
message()
```

Note: press ctrl + c ---> will get above exception.

Output:

Hello

Hello

...

...

KeyboardInterrupt

ModuleNotFoundError:

When ever we are importing a specific module if that module is not available then we will get this exception.

Example:

>>> import math

>>> import math123

ModuleNotFoundError: No module named 'math123'

ImportError:

- This exception is raised if using the import keyword to import a module fails.
- The ImportError is raised when an import statement has trouble successfully importing the specified module.
- Typically, such a problem is due to an invalid or incorrect path, which will raise a ModuleNotFoundError in Python 3.6 and newer versions.

Example:

```
>>> from  math  import  pow1
ImportError: cannot import name 'pow1' from 'math' (unknown location)
```

AttributeError:

- This exception is raised if a variable reference or assigning a value fails.

Example:

```
>>> lst = [10, 20, 30]
>>> lst.discard(10)
AttributeError: 'list' object has no attribute 'discard'
```

StopIteration:

- If the next() method returns null for an iterator, this exception is raised.
- Create a generator object with some range of values and access them using **next()** method.
- After accessing all values which are available in generator object using next() then it throws exception like **StopIteration**

Example:

```
>>> generator_object = (i for i in range(2))
>>> next(generator_object)
0
>>> next(generator_object)
1
>>> next(generator_object)
StopIteration
```

RuntimeWarning:

Example:

```
import numpy as np
print(np.sqrt(-1))
```

Output:

RuntimeWarning: invalid value encountered in sqrt
Nan

FloatingPointError:

- Python FloatingPointError is an error that occurs when a floating-point calculation is attempted that is not supported by the available hardware or software.

Example:

```
import numpy as np
with np.errstate(invalid='raise'):
   print(np.sqrt(-16))
```

Output:

FloatingPointError: invalid value encountered in sqrt

Q 1) How does Python handle an exception?

- Python handles exceptions using the try-except statement. The code that can raise an exception is placed and executed in the try block while the except block holds the code that will handle the exceptions if any arises.

Q 2) What is raising an exception in Python?

- Whenever the Python interpreter encounters an invalid code, it raises an exception, which is Python's own way to tell us that something unexpected happened.
- We can intentionally raise exceptions using the raise statement.

Q 3) How does Python handle multiple exceptions?

- Python handles multiple exceptions using either a single except block or multiple except blocks.
- For a single block, the exceptions are passed as a tuple: **except(Exception1, Exception2,..,ExceptionN)** and Python checks for a match from right to left. In this case, the same action is taken for each exception.
- Another way to catch all exceptions is to leave out the name of the exception after the except keyword.

```
except:      # handle all exceptions here
    pass
```

- This way, you can take common actions for all Exceptions.

Note: The second way is to use an except block for each exception:

```
except Exception1:
    # code to handle Exception1 goes here
except Exception2:
    # code to handle Exception2 goes here
except ExceptionN:
    # code to handle ExceptionN goes here
```

- This way, you can take separate actions for each Exception.

Q 4) Why is Exception handling important in Python ?

- The benefit of handling exceptions in Python is that we can create robust, clean and error-free applications.
- We won't want our production code to crash due to some errors, so we handle the errors and keep our application up and running.

Q 5) How do you ignore an exception in Python?

- To ignore an exception in Python, use the pass keyword in the except block. Let's say we want to ignore the **ValueError** exception. We will do it this way.

```
except ValueError:
    pass
```

- Unless you know what you are doing, it is bad practice to ignore exceptions. At least, inform the user about all potential errors.

Exception : Cause of Error

AssertionError :

- Raised when assert statement fails.

AttributeError:

- Raised when attribute assignment or reference fails.

FloatingPointError :

- Raised when a floating point operation fails.

GeneratorExit :

- Raise when a generator's close() method is called.

ImportError :

- Raised when the imported module is not found.

IndexError :

- Raised when index of a sequence is out of range.

KeyError :

- Raised when a key is not found in a dictionary.

KeyboardInterrupt :

- Raised when the user hits interrupt key (Ctrl+c or delete).

MemoryError :

- Raised when an operation runs out of memory.

NameError :

- Raised when a variable is not found in local or global scope.

NotImplementedError :

- Raised by abstract methods.

OSError :

- Raised when system operation causes system related error.

OverflowError :

- Raised when result of an arithmetic operation is too large to be represented.

ReferenceError :

- Raised when a weak reference proxy is used to access a garbage collected referent.

RuntimeError :

- Raised when an error does not fall under any other category.

StopIteration :

- Raised by next() function to indicate that there is no further item to be returned by iterator.

SyntaxError :

- Raised by parser when syntax error is encountered.

IndentationError :

- Raised when there is incorrect indentation.

TabError :

- Raised when indentation consists of inconsistent tabs and spaces.

SystemError :

- Raised when interpreter detects internal error.

SystemExit :

- Raised by sys.exit() function.

TypeError :

- Raised when a function or operation is applied to an object of incorrect type.

UnboundLocalError :

- Raised when a reference is made to a local variable in a function or method, but no value has been bound to that variable.

UnicodeError :

- Raised when a Unicode-related encoding or decoding error occurs.

UnicodeEncodeError :

- Raised when a Unicode-related error occurs during encoding.

UnicodeTranslateError :

- Raised when a Unicode-related error occurs during translating.

ValueError :

- Raised when a function gets argument of correct type but improper value.

ZeroDivisionError :

- Raised when second operand of division or modulo operation is zero.
- We can handle these built-in and user-defined exceptions in Python using try, except and finally statements.