Exception Name (..)

Set:

Sets are used to store multiple items in a single variable. A set is a collection which is *unordered*, *unchangeable**, and *unindexed*. Set *items* are unchangeable, but we can remove items and add new items.

```
thisset = {"apple", "banana", "cherry"}
print(thisset)
```

- Sets are unordered, so you cannot be sure in which order the items will appear.
- We cannot change the item after set has been created.
- It does not allow duplicate values.

Dictionary:

Dictionaries are used to store data values in key. A dictionary is a collection which is ordered*, changeable and does not allow duplicates. Dictionaries are written with curly brackets, and have keys and values:

```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
print(thisdict)
```

- In the latest versions of python compilers, dictionaries are ordered.
- We can change, add or remove items after the dictionary has been created.
- Dictionaries cannot have two items with the same key.

Exception Handling:

If you have some *suspicious* code that may raise an exception, you can defend your program by placing the suspicious code in a **try:** block. After the try: block, include

an **except:** statement, followed by a block of code which handles the problem as elegantly as possible.

Syntax

Here is simple syntax of *try....except...else* blocks –

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

A list of various exceptions are as follows:

Exception Name	Exception Description
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.
AttributeError	Raised in case of failure of attribute reference or assignment.
EOFError	Raised when there is no input from either the raw_input() or input() function and the end of file is reached.
ImportError	Raised when an import statement fails.
KeyboardInterrupt	Raised when the user interrupts program execution, usually by pressing Ctrl+c.

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.
UnboundLocalError	Raised when trying to access a local variable in a function or method but no value has been assigned to it.
EnvironmentError	Base class for all exceptions that occur outside the Python environment.
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.
IndentationError	Raised when indentation is not specified properly.
SystemError	Raised when the interpreter finds an internal problem, but when this error is encountered the Python interpreter does not exit.
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.
NotImplementedError	Raised when an abstract method that needs to be implemented in an inherited class is not actually implemented.

1. Predict the output:

```
try:
   f = open("file2.txt", "r+")
   f.write("Hello All")
   f.write("Welcome to BU")
except IOError as IO_Error:
   print (IO_Error)
else:
   print ("File updated successfully !!")
```

Note: if file2.txt doesn't exist

Sol: `[Errno 2] No such file or directory: 'file2.txt'`

2. Predict the output:

```
def fun(arg1):
   try:
      return int(arg1)
   except ValueError as Val_err:
      print (Val_err)
   except IOError as IO_error:
    print (IO_error)
   except TypeError as Type_error:
    print(Type_error)
fun("abc")
```

Sol: `invalid literal for int() with base 10: 'abc'`
Explanation: Value Error

3. Predict the output:

```
try:
    a = 100 / 0
    print (a)
except ZeroDivisionError as ZeroDivisionError:
        print (ZeroDivisionError)
else:
    print ("Success, no error!")
```

Sol: `division by zero`

4. Predict the output:

```
#File1.txt:
#    Bennett University
#    Greater Noida

import sys

try:
    f = open('File1.txt')
    s = f.readline()
    i = int(s.strip())
    print(i)
except OSError as err:
    print(err)
except ValueError as ValueError:
    print(ValueError)
except BaseException as BaseException_err:
    print(BaseException_err)
    raise
```

Sol: `invalid literal for int() with base 10: 'Bennett University'`
Explanation: Value Error

5. Predict the output:

```
import sys

try:
    f = open('File2.txt')
    s = f.readline()
    i = int(s.strip())
    print(i)
except OSError as err:
    print(err)
except ValueError as ValueError:
    print(ValueError)
except BaseException as BaseException:
    print(BaseException)
    raiseNote: if file2.txt doesn't exist
```

Sol: `[Errno 2] No such file or directory: 'File2.txt'`
Explanation: OSError

6. Predict the output, and explain in detail:

```
executed = False
while not executed:
    try:
        a = int(input('first number --> '))
        b = int(input('second number --> '))
        z = a / b
        print(z)
        executed = True
    except ArithmeticError as arithmeticError:
        print(arithmeticError)
    except ValueError as valueError:
        print(valueError)
    except Exception as exception:
        print(exception)
```

Note:
```
a = 12
b = 0
```

Sol:
```
first number --> 12
second number --> 0
division by zero
```
Explanation: we are dividing until correct data is given

7. Predict the output:

```
a = ['apple', 'banana', 'cherry']
try:
    print("element_1 = %d" %(a[1]))
    print("element_2 = %d" %(a[3]))
except IndexError as indx_err:
    print(indx_err)
except ValueError as Val_err:
    print (Val_err)
except IOError as Io_err:
    print (Io_err)
except TypeError as Typ_err:
    print(Typ_err)
```

Sol:
```
%d format: a number is required, not str
```

8. Predict the output:

```
try:
    a = [1, 2, 3]
    print (a[3])
except LookupError as error:
    print (error)
else:
    print ("Success")
```

Sol: `list index out of range`

9. Predict the output:

```
import math
def Fun1(a, b, c):
    try:
        assert a != 0, "Not a quadratic equation as coefficient of x ^ 2
can't be 0"
        D = (b * b - 4 * a*c)
        assert D>= 0, "Roots are imaginary"
        r1 = (-b + math.sqrt(D))/(2 * a)
        r2 = (-b - math.sqrt(D))/(2 * a)
        print("Roots of the quadratic equation are :", r1, "", r2)
    except AssertionError as msg:
        print(msg)
Fun1(-1, 5, -6)
Fun1(1, 1, 6)
Fun1(2, 12, 18)
```

Sol.

```
Roots of the quadratic equation are : 2.0  3.0
Roots are imaginary
Roots of the quadratic equation are : -3.0  -3.0
```

10. Predict the output, explain in detail:

```
def fun1():
    try:
        geek = "BU"
        return BU
    except NameError as Name_Error:
        return Name_Error
print(fun1())
```

Sol. `name 'BU' is not defined`

11. Predict the output:

```
import sys
try:
    vowels = ['a', 'e', 'i', 'o', 'u']
    i = iter(vowels)
    print (next(i))
    print (i.next())
except Exception as e:
    print(e)
```

Sol.

```
a
'list_iterator' object has no attribute 'next'
```

12. Predict the output:

```
try:
    f = open("File1.txt",'r')
    try:
        f.write("Hello Bennett University")
        print(f.read())
    except:
        print("Something went wrong when writing to the file")
    finally:
        f.close()
except:
    print("Something went wrong when opening the file")
```

Sol.

```
Something went wrong when writing to the file
```

13. Predict the output:

```
value = [1, 2, 3, 4]
data = 0
try:
    data = value[5]
except IndexError:
    print('it is IndexError')
except:
    print('again IndexError')
finally:
    print('it is finally IndexError')

data = 10
try:
    data = data/0
except ZeroDivisionError:
    print('Zero Division Error')
finally:
    print('it is ZeroDivisionError')
```

Sol:

```
it is IndexError
it is finally IndexError
Zero Division Error
it is ZeroDivisionError
```

14. Predict the output:

```
def fun(n):
    return n.append(n)

try:
    f1 = fun(2)
    print(f1)
except AttributeError as e:
    print('int object has no attribute append')
```

Sol.

```
  int object has no attribute append
```

15. Predict the output:

```
import operator
d = {1: 2, 3: 4, 4: 3, 2: 1, 0: 0}
print('dictionary : ',d)
sorted_d = sorted(d.items(), key=operator.itemgetter(1))
print('values are : ',sorted_d)
sorted_d = dict( sorted(d.items(),
key=operator.itemgetter(1),reverse=True))
print('values are : ',sorted_d)
```

Sol:

```
dictionary :  {1: 2, 3: 4, 4: 3, 2: 1, 0: 0}
values are :  [(0, 0), (2, 1), (1, 2), (4, 3), (3, 4)]
values are :  {3: 4, 4: 3, 1: 2, 2: 1, 0: 0}
```

16. Predict the output:

```
x = int(input())
try:
    import math
    print(math.exp(x))
except OverflowError as OverflowError:
        print (OverflowError)
else:
    print ("Success, no error!")
```

Note: `x = 4`

Sol.

```
4
54.598150033144236
Success, no error!
```

`X = 2000`

Sol.

```
2000
math range error
```

Exercise:

```
# Creating a set
Days = (["Mon","Tue","Wed","Thu","Fri","Sat","Sun"])
Months = {"Jan","Feb","Mar"}
Dates = {21,22,17}

#Adding Items to a Set
Days = set(["Mon","Tue","Wed","Thu","Fri","Sat"])
Days.add("Sun")
```

```
#Removing Item from a Set
Days=set(["Mon","Tue","Wed","Thu","Fri","Sat"])
Days.discard("Sun")
```

```
#Union of Sets
DaysA = set(["Mon","Tue","Wed"])
DaysB = set(["Wed","Thu","Fri","Sat","Sun"])
AllDays = DaysA|DaysB
```

```
#Intersection of Sets
DaysA = set(["Mon","Tue","Wed"])
DaysB = set(["Wed","Thu","Fri","Sat","Sun"])
AllDays = DaysA & DaysB
```

```
#Difference of Sets
DaysA = set(["Mon","Tue","Wed"])
DaysB = set(["Wed","Thu","Fri","Sat","Sun"])
AllDays = DaysA - DaysB


#Compare Sets
DaysA = set(["Mon","Tue","Wed"])
DaysB = set(["Mon","Tue","Wed","Thu","Fri","Sat","Sun"])
SubsetRes = DaysA <= DaysB
SupersetRes = DaysB >= DaysA
```