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("Bennett University")
   f.write("Greater Noida")
except IOError:
   print ("Error: can\'t find file or read data")
else:
   print ("File updated successfully !!")

Note: if file2.txt doesn't exist
```

Sol: Error: can't find file or read data

2. Predict the output:

```
def fun(arg1):
   try:
      return int(arg1)
   except ValueError:
      print ("The argument does not contain numbers\n")
   except IOError:
    print ("Error: IOErrors occurs")
   except TypeError:
    print("Error: TypeError occurs")

fun("abc")
```

Sol: The argument does not contain numbers

3. Predict the output:

```
import math
x = int(input('Please enter a number: \n'))
try:
    print(f'Square Root of {x} is {math.sqrt(x)}')
except ValueError:
    print(f'Oops!  That was no valid number.  Try again...')
```

Note: x = -10

Sol: Oops! That was no valid number. Try again...

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("OS error: {0}".format(err))
except ValueError:
    print("Could not convert data to an integer.")
except BaseException as err:
    print(f"Unexpected {err=}, {type(err)=}")
    raise
```

Sol: `Could not convert data to an integer.`

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("OS error: {0}".format(err))
except ValueError:
    print("Could not convert data to an integer.")
except BaseException as err:
    print(f"Unexpected {err=}, {type(err)=}")
    raise
```

Note: if file2.txt doesn't exist

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

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:

```
a = (22, 20, 24, 26)
try:
    print("element_1 = %d" %(a[2]))
    print("element_2 = %d" %(a[4]))
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:

```
element_1 = 24
list index out of range
```

9. Predict the output:

```
try:
    a = int(input())
    if a < 4:
        b = a/(a-2)
    print("Value of b = ", b)
except ZeroDivisionError as Zero_div_Error:
    print(Zero_div_Error)
except ValueError as Val_Error:
    print (Val_Error)
```

Note: `a = 2`

Sol:

```
division by zero
```

10. Predict the output, explain in detail:

```
name = 'Bennett University'
try:
    print(name[25])
except AssertionError as Asser_err:
   print(Asser_err)
except (EnvironmentError, SyntaxError, NameError) as E:
    print(E)
except Exception as excp:
    print(excp)
else:
        pass
finally:
    pass

print('This will be printed.')
```

Sol:

```
string index out of range
This will be printed.
```

11. Predict the output:

```
try:
    x = 1
    y = 0
    assert y != 0, "Invalid Operation"
    print(x / y)

except AssertionError as msg:
    print(msg)
```

Sol.

```
    Invalid Operation
```

12. Predict the output:

```
import sys

randomList = ['BU', 0, 1]

for entry in randomList:
    try:
        print("The evalue is", entry)
        r = 1/int(entry)
        break
    except Exception as Exp_err:
        print(Exp_err)
```

Sol:

```
The evalue is BU
invalid literal for int() with base 10: 'BU'
The evalue is 0
division by zero
The evalue is 1
```

13. Predict the output:

```
Age = {'Ram' : 10, 'John' : 15, 'Aryan' : 25}
item = input('Get your age: ')

try:
    print(f'Age of {item} is {Age[item]}')
except KeyError:
    print(f'Age of {item} is not known')
```

item = `Tony`

Sol: = `Age of Tony is not known`

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:

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

Note: `x = 2`

Sol.

```
7.38905609893065
Success, no error!
```

```
X = 1000
```

Sol.

```
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
```