Exception Name (..)

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("file1", "r")
   f.write("Hello friends")
except IOError:
   print ("Error: can\'t find file or read data")
else:
   print ("Written content in the file successfully")
```

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

2. Predict the output:

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

convert("abc")
```

Sol: `The argument does not contain numbers`

3. Predict the output:

```
def divide(x, y):
    try:
        result = x / y
    except ZeroDivisionError:
        print("division by zero!")
    except ValueError:
        print ("it is value type error\n")
    else:
        print("result", result)
    finally:
        print("executing finally")

divide(2, 0)
```

Sol:

```
division by zero!
executing finally
```

4. Predict the output:

```
a = [2, 3, 4]
try:
    print("element_1 = %d" %(a[1]))
    print("element_2 = %d" %(a[3]))

except IndexError:
    print("out of index occurred")
except ValueError:
    print ("The argument does not contain numbers\n")
except IOError:
    print ("Error: IOErrors occurs")
except TypeError:
    print("Error: TypeError occurs")
```

Sol:

```
element_1 = 3
out of index occurred
```

5. Predict the output:

```
try :
    a = 3
    #a = 4   find for this also
    if a < 4 :
        b = a/(a-3)
    print("Value of b = ", b)
except(ZeroDivisionError, NameError):
    print("\nzero or name error Occurred")
except ValueError:
    print ("value error occured\n")
```

Sol:

```
zero or name error Occurred
```

6. Predict the output:

```
import sys

randomList = ['b', 0, 1]

for entry in randomList:
    try:
        print("The evalue is", entry)
        r = 1/int(entry)
        break
    except Exception as e:
        print("Oops!", e.__class__, "occurred.")
        print("Oops!", e, "occurred.")
```

Sol:

```
The evalue is b
Oops! <class 'ValueError'> occurred.
Oops! invalid literal for int() with base 10: 'b' occurred.
```

```
The evalue is 0
Oops! <class 'ZeroDivisionError'> occurred.
Oops! division by zero occurred.
The evalue is 1
```

7. 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
```

Sets(...)

```
# 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
```

8. Predict the output:

```
nset = set([0, 1, 2, 3, 4, 5])
for n in nset:
  print(n)
```

Sol:

```
0
1
2
3
4
5
```

9. Predict the output:

```
cset = set()
cset.add("Red")
print(cset)
cset.update(["Blue", "Green"])
print(cset)
```

Sol:

```
{'Red'}
{'Green', 'Red', 'Blue'}
```

10. Predict the output:

```
nset1 = set([0, 1, 3, 4, 5])
nset2 = {4,5,6}
nset1.pop()
nset1.add(8)
nset3=nset1-nset2
print(nset3)
```

Sol:

```
{8, 1, 3}
```

11. Predict the output:

```
sn1 = {1,2,3}
sn2 = {4,5,6}
sn3 = {3}
print(sn1.isdisjoint(sn2))
print(sn1.isdisjoint(sn3))
sn4 = sn1 & sn3
print(sn4)
sn5 = sn1|sn4
print(sn5)
```

Sol:

```
True
False
{3}
{1, 2, 3}
```

12. Write a Python code for checking if two given sets have no common elements.

Sol:

```
x = {1,2,3,4}
y = {4,5,6,7}
z = {8}
print(x.isdisjoint(y))
print(z.isdisjoint(x))
print(y.isdisjoint(z))

Output:
False
True
True
```

13. Write a Python code for checking if a given set is superset of itself and superset of another given set.

Sol:

```
num1 = {1, 2, 3, 4, 5, 7}
```

```
num2 = {2, 4}
num3 = {2, 4}
print(num1>num2)
print(num2>num3)
print(num3>num2)
```

Output:

```
True
False
False
```

Dictionary(…)

14. Predict the output of the following code.

```
test_str = 'bu_is_best_for_btech'
print("original string: "+ str(test_str))
delim = "_"
temp = test_str.split(delim)
res = dict()
for idx, ele in enumerate(temp):
    res[idx] = ele
print("after splits: " + str(res))
```

Sol:

```
original string: bu_is_best_for_btech
after splits: {0: 'bu', 1: 'is', 2: 'best', 3: 'for', 4: 'btech'}
```

15. Predict the output of the following code.

```
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}
```

In [ ]:

16. Predict the output of the following code.

```
d=dict()
for x in range(1,16):
    d[x]=x**2
print(d)
```

Sol:

```
{1: 1, 2: 4, 3: 9, 4: 16, 5: 25, 6: 36, 7: 49, 8: 64, 9: 81, 10: 100, 11:
121, 12: 144, 13: 169, 14: 196, 15: 225}
```

17. Predict the output of the following code.

```
d1 = {'a': 100, 'b': 200}
d2 = {'x': 300, 'y': 200}
d = d1.copy()
d.update(d2)
print(d)
```

Sol:

```
{'a': 100, 'b': 200, 'x': 300, 'y': 200}
```

18. Predict the output of the following code.

```
keys = ['red', 'green', 'blue']
values = ['#FF0000','#008000', '#0000FF']
cdictionary = dict(zip(keys, values))
print(cdictionary)
```

Sol:

```
{'red': '#FF0000', 'green': '#008000', 'blue': '#0000FF'}
```

19. Predict the output of the following code.

```
mdict = {'x':500, 'y':5874, 'z': 560}
kmax = max(my_dict.keys(), key=(lambda k: my_dict[k]))
kmin = min(my_dict.keys(), key=(lambda k: my_dict[k]))
print('MValue: ',my_dict[kmax])
print('MValue: ',my_dict[kmin])
```

Sol:

```
MValue:  5874
MValue:  500
```

20. Write a Python code for combining two dictionary adding values for common keys.

```
Sample Input:
d1 = {'a': 100, 'b': 200, 'c':300}
d2 = {'a': 300, 'b': 200, 'd':400}
```

Sample output:
Counter({'a': 400, 'b': 400, 'd': 400, 'c': 300})

Sol:

```
from collections import Counter
d1 = {'a': 100, 'b': 200, 'c':300}
d2 = {'a': 300, 'b': 200, 'd':400}
d = Counter(d1) + Counter(d2)
print(d)
```