Predict the output of the following code for these function calls:

(a) divide(2, 1) (b) divide(2, 0) (c) divide("2", "1")

```
def divide(x, y):
    try:
        result = x/y
    except ZeroDivisionError:
        print ("division by zero!")
    else:
        print ("result is", result)
    finally:
        print ("executing finally clause")
```

Solution.

(a) divide(2, 1) result is 2.0
executing finally clause
(b) divide(2, 0) division by zero!
executing finally clause
(c) divide("2", "1")
executing finally clause
Traceback (most recent call last):
TypeError: unsupported operand type(s) for /: 'str' and 'str'

1 Marks Question:

1. Errors resulting out of violation of programming language's grammar rules are known as:
   (a) Compile time error (b) Logical error (c) Runtime error (d) Exception
2. An unexpected event that occurs during runtime and causes program disruption, is called:
   (a) Compile time error (b) Logical error (c) Runtime error (d) Exception
3. Which of the following keywords are not specific to exception handling ?
   (a) try (b) except (c) finally (d) else
4. Which of the following blocks is a 'must-execute' block ?
   (a) try (b) except (c) finally (d) else
5. Which keyword is used to force an exception ?
   (a) try (b) except (c) raise (d) finally

Python raise Keyword is used to raise exceptions or errors. The raise keyword raises an error and stops the control flow of the program. It is used to bring up the current exception in an exception handler so that it can be handled further up the call stack.

Syntax of the raise keyword :

```
raise  {name_of_ the_ exception_class}
```

The basic way to raise an error is :

```
raise Exception("user text")
```

Example 1:

```
a = 5

if a % 2 != 0:
    raise Exception("The number shouldn't be an odd integer")
```

Output:

```
Traceback (most recent call last):
  File "/home/3dd59d932258303ca09ee7c2e500e499.py", line 4, in <module>
    raise Exception("The number shouldn't be an odd integer")
Exception: The number shouldn't be an odd integer
```

Example 2:

```
s = 'apple'

try:
    num = int(s)
except ValueError:
    raise ValueError("String can't be changed into integer")
```

Output:

```
Traceback (most recent call last):
  File "/home/a095919ad2ab2bfcbde6c9b994b0c705.py", line 3, in <module>
    num = int(s)
ValueError: invalid literal for int() with base 10: 'apple'

During handling of the above exception, another exception occurred:

Traceback (most recent call last):
  File "/home/a095919ad2ab2bfcbde6c9b994b0c705.py", line 5, in <module>
    raise ValueError("String can't be changed into integer")
ValueError: String can't be changed into integer
```