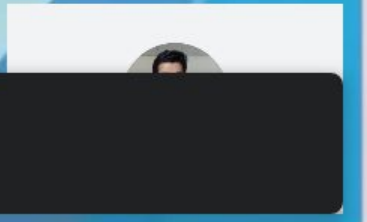
REC

Exceptions

- Exception handling is one of the most important feature of Python programming language that allows us to handle the errors caused by exceptions.
- Even if a statement or expression is syntactically correct, it may cause an error when an attempt is made to execute it.
- Errors detected during execution are called ***exceptions***.
- An exception is an unwanted event that interrupts the normal flow of the program.
- When an exception occurs in the program, execution gets terminated. In such cases, we get a system-generated error message. However, these exceptions can be handled in Python.
- By handling the exceptions, we can provide a meaningful message to the user about the issue rather than a system-generated message, which may not be understandable to the user.



Shivanshu joined

REC

Exceptions can be either built-in exceptions or user-defined exceptions.

```
>>> 10 * (1/0)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ZeroDivisionError: division by zero
>>> 4 + spam*3
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'spam' is not defined
>>> '2' + 2
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: Can't convert 'int' object to str implicitly
```

You

REC

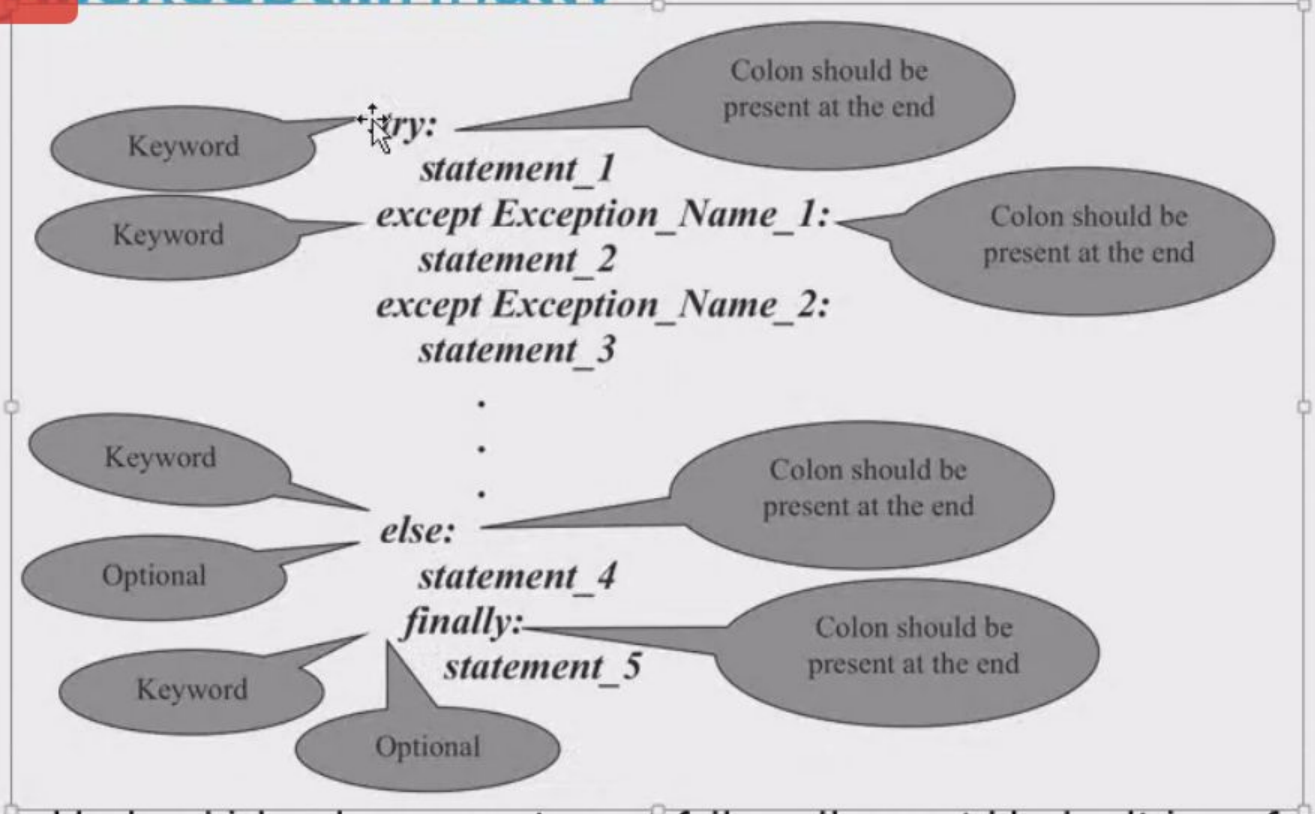
Exception Handling Using try...except...finally

- Handling of exception ensures that the flow of the program does not get interrupted when an exception occurs which is done by trapping run-time errors.
- Handling of exceptions results in the execution of all the statements in the program.
- Run-time errors are those errors that occur during the execution of the program.
- These errors are not detected by the Python interpreter, because the code is syntactically correct.

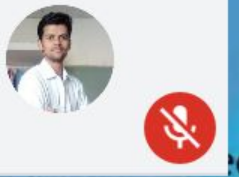
You

REC

...except...finally



The else block, which, when present, must follow all except blocks. It is useful for
executed if the try block does not raise an exception.
A finally block is always executed before leaving the try statement, whether an exc



REC

Program to check for *ValueError* Exception

```
while True:
    try:
        number = int(input("Please enter a number: "))
        print(f"The number you have entered is {number}")
        break
    except ValueError:
        print("Oops! That was no valid number. Try again…")
```

```
Please enter a number: g
Oops! That was no valid number. Try again…
Please enter a number: 4
The number you have entered is 4
```

You

REC

Program

- Program to Check for ZeroDivisionError Exception

```
1  x = int(input("Enter value for x: "))
2  y = int(input("Enter value for y: "))
3  try:
4      result = x / y
5  except ZeroDivisionError:
6      print("Division by zero!")
7  else:
8      print(f"Result is {result}")
9  finally:
10     print("Executing finally clause")
```

```
Enter value for x: 8
Enter value for y: 0
Division by zero!
Executing finally clause
```

```
Enter value for x: 4
Enter value for y: 2
Result is 2.0
Executing finally clause
```

The *else* block gets executed if the *try* block does not raise an exception.

You

Try except finally

```
#compile time error- syntax error,  #logical error-getting wrong answers, #runtime error-6/0-dividebyzero error
a=5
b=0
try:
    print("resource open")
    print(a/b)
    k=int(input("Enter "))
    print(k)
except ZeroDivisionError as e:
    print(" cant divide by zero",e)
except ValueError as v:
    print(" invalid input",v)
except Exception as p:
    print(" oops: something wrong",p)
finally:
    print("resource closed")
print("thanks")
```

You

Write a Program Which Repeatedly Reads Numbers Until the User Enters 'done'. Once 'done' is Entered, Print Out the Total, Count, and Average of the Numbers. If the User Enters Anything Other Than a Number, Detect Their Mistake Using try and except and Print an Error Message and Skip to the Next Number

```python
total = 0
count = 0
while True:
    num = input("Enter a number: ")
    if count != 0 and num == 'done':
        print(f"Sum of all the entered numbers is {total}")
        print(f"Count of total numbers entered {count}")
        print(f"Average is {total / count}")
        break
    else:
        try:
            total += float(num)
        except:
            print("Invalid input")
            continue
        count += 1
```

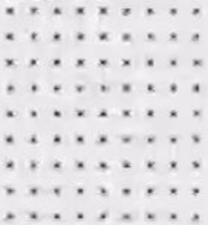
```
Enter a number: 1
Enter a number: 2
Enter a number: 3
Enter a number: 4
Enter a number: 5
Enter a number: done
Sum of all the entered numbers is 15.0
Count of total numbers entered 5
Average is 3.0
```


REC

Patterns

Enter the number of rows 9

Pattern-1



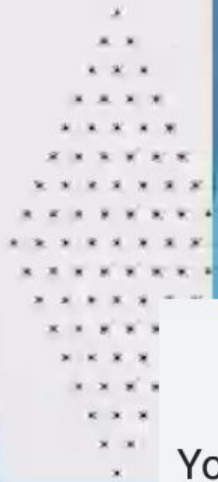
Pattern-2

Pattern-3

Pattern-4

Pattern-5

Pattern-6



Pattern-7

Pattern-8

Pattern-9

Pattern-10

You