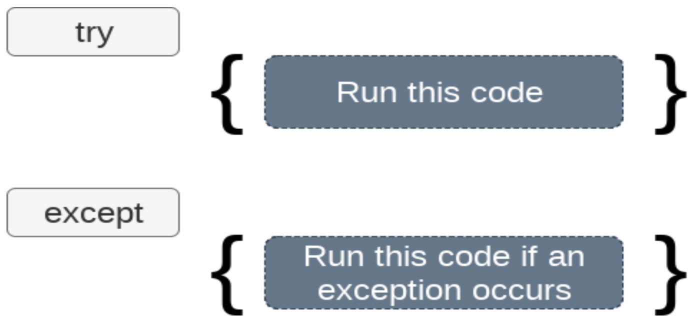
Exception Handling

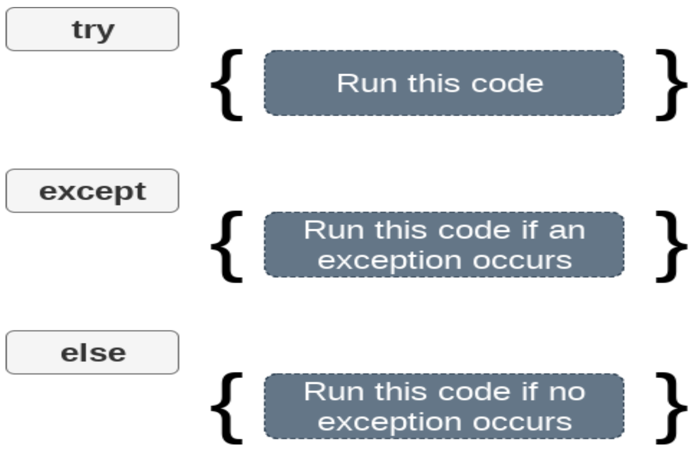
- An exception can be defined as an abnormal condition in a program resulting in the disruption inthe flow of the program.
- Whenever an exception occurs, the program halts the execution, and thus the further code is not executed.
- Python provides us with the way to handle the Exception so that the other part of the code canbe executed without any disruption.
- **Some Common Exceptions:**
    - ZeroDivisionError: Occurs when a number is divided by zero.
    - NameError: It occurs when a name is not found. It may be local or global.
    - IndentationError: If incorrect indentation is given.
    - IOError: It occurs when Input Output operation fails.
    - EOFError: It occurs when the end of the file is reached, and yet operations are being performed
- If the python program contains suspicious code that may **throw the exception**, we must place that code in the **try block**.
- The try block must be followed with the **except statement** which contains a block of code that will be executed if there is some exception in the try block.



- **Syntax:**
    - try:
    -     #block of code
    - except Exception1:
    -     #block of code

- except Exception2:
-     #block of code
- #other code

- We can also use the else statement with the try-except statement in which, we can place the code which will be executed in the scenario if no exception occurs in the try block.


try
Run this code
except
Run this code if an exception occurs
else
Run this code if no exception occurs

- **Syntax:**

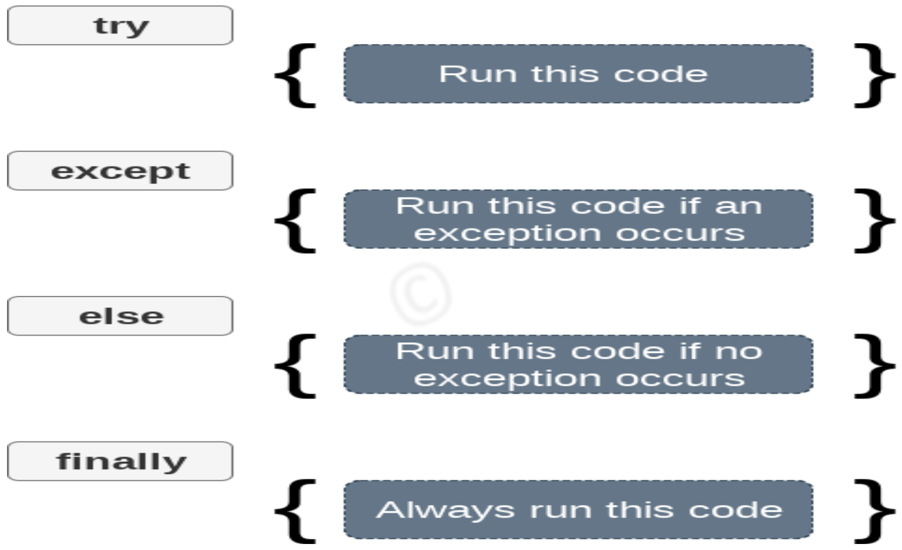
  - try:
  -     #block of code
  - Except Exception 1:
  -     #block of code
  - else:
  -     #this code executes if no except block is executed

- **The except statement with no exception**
  - Python provides the flexibility not to specify the name of exception with the except statement.
    - try:
    -     a = int(input("Enter a number:"))
    -     b = int(input("Enter a number:"))
    -     print(a/b)
    - except:
    -     print("Can't Divided by zero.")
    - else:
    -     print("Hiii I am else block..")

- **Output:**

```
-  Enter a number:1
-  Enter a number:0
-  Can't Divided by zero.
```

- **<u>Points to remember</u>**
  - Python facilitates us to not specify the exception with the except statement.
  - We can **declare multiple exceptions** in the except statement since the try block may contain the statements which throw the different type of exceptions.
  - ***We can also specify an else block along with the try-except statement which will be executed if no exception is raised in the try block***.
  - The statements that don't throw the exception should be placed inside the else block.
- **<u>The finally block</u>**
  - We can use the **finally block with the try block** in which, we can pace the important code which must be **executed before the try statement throws an exception**.


try
{ Run this code }
except
{ Run this code if an exception occurs }
else
{ Run this code if no exception occurs }
finally
{ Always run this code }

  - **Syntax:**

```
-  try:
-      #block of code
-  Except Exception 1:
-       #block of code
-  else:
```

- #this code executes if no except block is executed
- finally:
- # block of code
- # this will always be executed

- **Raising exceptions**
  - An exception can be **raised** by using the **raise clause** in python.
  - The syntax to use the raise statement is given below.
    - raise Exception_class,<value>
  - To raise an exception, **raise statement** is used. The exception class name follows it.
  - An exception can be provided with a value that can be given in the parenthesis.
  - To access the value "as" keyword is used. "e" is used as a reference variable which stores the value of the exception
  - **Example 1:**

```
- try:
-     age = int(input("Enter your age="))
-     if age<18:
-         raise ValueError
-     else:
-         print("The age is valid.")
- except ValueError:
-     print("The age is not valid.")
```

  - **Output:**

```
- Enter your age=11
- The age is not valid.
```

  - **Example 2:**

```
- try:
-     a = int(input("Enter a number:"))
-     b = int(input("Enter a number:"))
-     if b is 0:
-         raise ZeroDivisionError
-     elif b < 1:
```

```
-           raise ArithmeticError
-       else:
-           print(a/b)
-
-   except ZeroDivisionError:
-       print("Denominator is zero")
-
-   except ArithmeticError:
-       print("Denominator is negative")
```

○ **Output:**

```
-   Enter a number:12
-   Enter a number:-2
-   Denominator is negative
```

○ **Example 3** Raise multiple Exception:

```
-   try:
-       div = 3 / 0
-       print(div)
-
-       l1 = [10,20,30,40]
-       l1[7] = 90
-       print(l1)
-
-   except ZeroDivisionError as e:
-       print(e)
-
-   except IndexError as e:
-       print(e)
-   else:
-       print("No exception.")
-
-   finally:
```

        - print("Always execute....")
    - **Output:**
        ```
        - division by zero
        - Always execute....
        ```
- We can use raise to throw an exception if a condition occurs.
- The statement can be complemented with a custom exception.
    - **Syntax:**
        - raise <Exception name>
        - raise ZeroDivisionError
        - **Mainly we use it for Custom or User defined Exception**
- **<u>Custom Exception</u>**
    - Python has many built in exceptions which forces your program to output an error when something in it goes wrong.
    - However you may need to create custom exception that serves your purpose
    - In python users can define such exceptions by creating a new class.
    - This exception class has to be derived directly or indirectly from Exception class
-