Inheritance
And
Exception Handling

Inheritance

Inheritance is the capability of one class to derive or inherit the properties from some another class.

It provides **reusability** of a code. We don’t have to write the same code again and again. Also, it allows us to add more features to a class without modifying it.

If class B inherits from another class A, then all the subclasses of B would automatically inherit from class A.

Parent class is the class being inherited from, also called base class.

Child class is the class that inherits from another class, also called derived class.

Parent class and Child class

```
class Profile:
  def __init__(self, name, address):
         self.name = name
         self.address = address



class Hr(Profile):
     def info(self):
    print(self.name, self.address)

x = Hr(“Ram”, “Nepal”)
x.info()
```

Example:

```
class Profile:
  def __init__(self, name, address):
    self.name = name
    self.address = address

  def info(self):
    print(self.name, self.address)

class Student(Profile):
  def __init__(self, name, address, country):
    Profile.__init__(self, name, address)
    self.country = country

  def hello(self):
    print("Hello, welcome", self.name,"from", self.address, "to ", self.country)

x = Student("John", "USA", "Nepal")
x.hello()
```

Python - public, private and protected Access

Public members (generally methods declared in a class) are accessible from outside the class. The object of the same class is required to invoke a public method.

Protected members of a class are accessible from within the class and are also available to its sub-classes. No other environment is permitted access to it. A variable that is protected can only be accessed by its own class and any classes derived from it.

Private members of a class are denied access from the environment outside the class. They can be handled only from within the class.

Public Attributes

```
class employee:
    def __init__(self, name, sal):
        self.name=name
        self.salary=sal
```

Protected Attributes

```
class employee:
    def __init__(self, name, sal):
        self._name=name  # protected attribute
        self._salary=sal # protected attribute
```

Private Attributes

```
class employee:
    def __init__(self, name, sal):
        self.__name=name  # private attribute
        self.__salary=sal # private attribute
```

Private

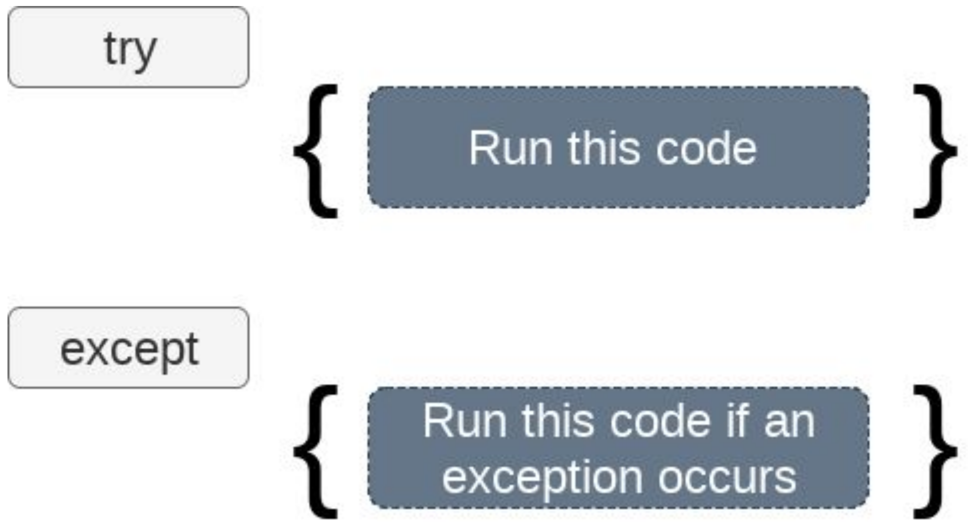
```
class Car():
    def __init__(self, name = 'Ram', age = 30, year = '1971', add = 'USA',
     color = 'black'):
        self.__name = name
        self._age = age
        self.__year = year
        self.__add = add
        self._color = color

    def move_forward(self, name):
        print(f"Hello My name is {self.__name}. {name} I am from {self.__add}.")

    def move_backward(self, age):
        print(f"Hello My name is {self.__name}.I am {self._age}. {age}")

mycar = Car()
print(mycar._color)              # changing to mycar.move_forward(100)
mycar.move_forward("Ajaya")
mycar.move_backward("50")
```

Exception Handling

- An exception is an error that happens during execution of a program. When that error occurs.
- Python generate an exception that can be handled, which avoids our program to crash.
- Raising an exception breaks current code execution and returns the exception back until it is handled.
- You can raise an exception in your own program by using the raise exception statement

Syntax:

```
try :
    #statements in try block
except :
    #executed when error in try block
```



Example:

```
try:
        age = int(input(“Age : ”))
        print(age)
except ValueError:
        print(“Invalid Value:Please enter age:”)
-------------------------------------------------------------------------------

try:
        age = int(input(“Age : ”))
        print(age)
except ValueError:
        print(“Invalid Value:Please enter age:”)
else:
        print(“No error”)
```

Examples

```
try:
    x=2
    y='0'
    print(x/y)
except:
    print('Some error occurred.')
```

Examples…

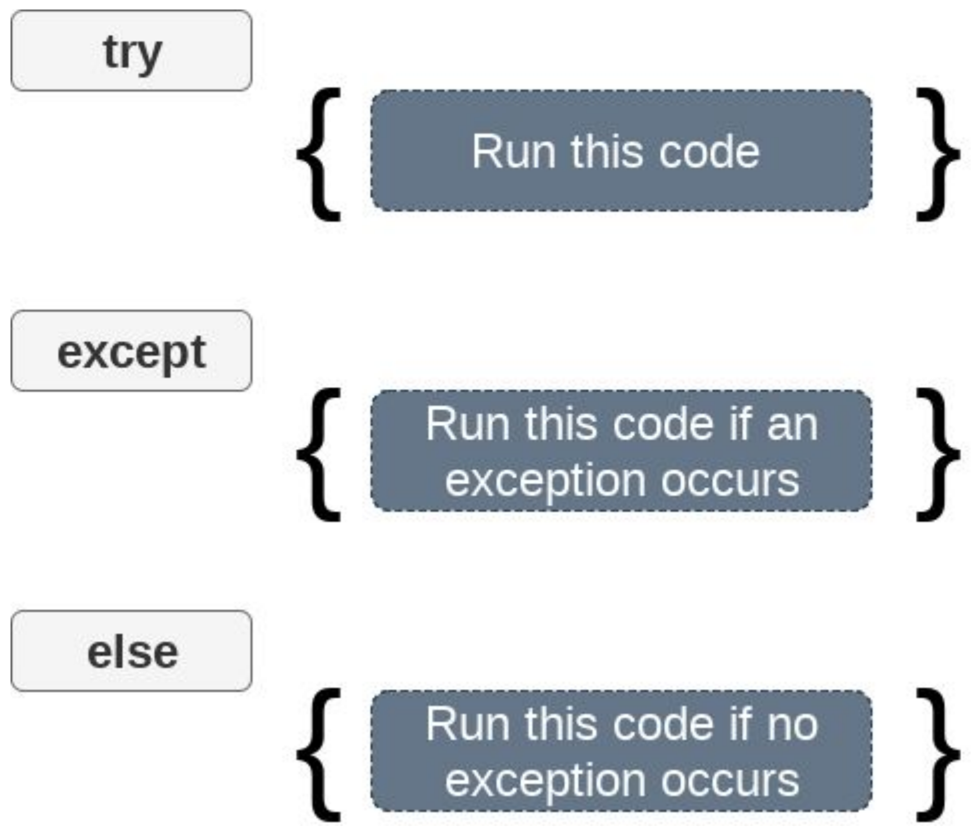
```
try:
    x=5
    y='0'
    print (x+y)

except TypeError:
    print('Unsupported operation')
```

Multiple except blocks

```
try:
    x=3
    y=0
    print (x/y)
except TypeError:
    print('Unsupported operation')
except ZeroDivisionError:
    print ('Division by zero not allowed')
```

Try,except,else

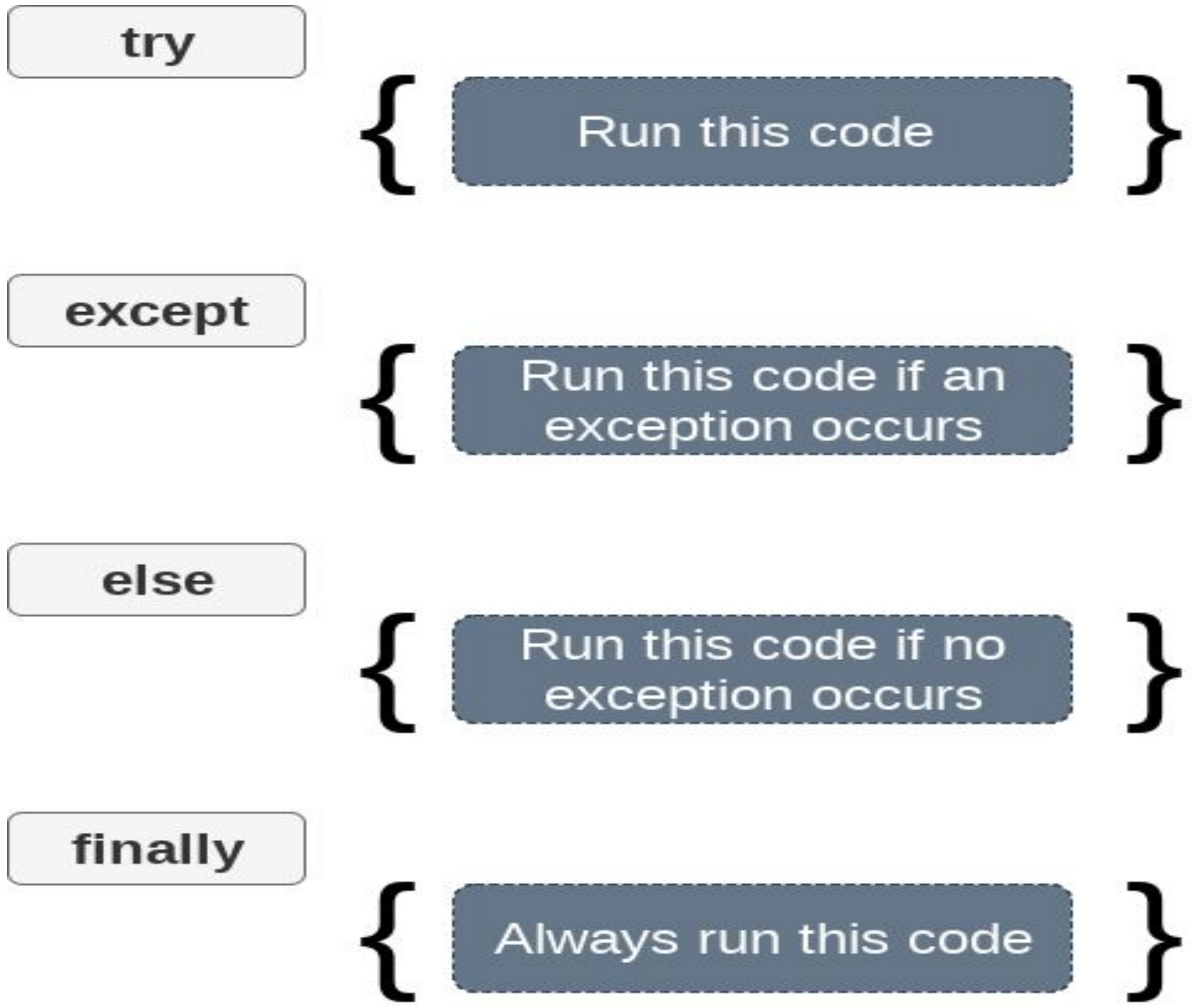
```
try:
    x = int(input("Enter x:"))
    y = int(input("Enter y:"))
    z= x/y
except:
    print("can't divide by zero")
else:
    print(“No error Further operation“,z)
```


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

Try,except,else

```
try:
    #this will throw an exception if the file doesn't exist.
    b = open(“book.txt","r")
except IOError:
    print("File not found")
else:
    print("The file opened successfully")
    b.close()
```

Try,except,else,finally


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

Try,finally,except

```
try:
    fileptr = open(“book.txt","r")
    try:
        fileptr.write("Hi I am good")
    finally:
        fileptr.close()
        print("file closed")
except:
    print("Error")
```

else and finally

```
try:
    #statements in try block
except:
    #executed when error in try block
else:
    #executed if try block is error-free
finally:
    #executed irrespective of exception occured or not
```

else and finally

```
try:
    print("try block")
    x=int(input('Enter a number: '))
    y=int(input('Enter another number: '))
    z=x/y
except ZeroDivisionError:
    print("except ZeroDivisionError block")
    print("Division by 0 not accepted")
else:
    print("else block")
    print("Division = ", z)
finally:
    print("finally block")
print ("Out of try, except, else and finally blocks." )
```

Try,raise,else,except

```
try:
    a = int(input("Enter a?"))
    b = int(input("Enter b?"))
    if b == 0:
        raise ZeroDivisionError
    else:
        print("a/b = ",a/b)
except ZeroDivisionError:
    print("The value of b can't be 0")
```

Python - Assert Statement

Python provides the **assert** statement to check if a given logical expression is true or false.

Program execution proceeds only if the expression is true and raises the **AssertionError** when it is false.

```
num=int(input('Enter a number: '))
assert num>=0
print('You entered: ', num)
```

Python - Assert Statement

```
try:
    num=int(input('Enter a number: '))
    assert(num >=0)
     print(num)
except AssertionError :
    print(“Error”)
```

Home Work

Discover Different exceptions handling and solve all those.