Exception Handling

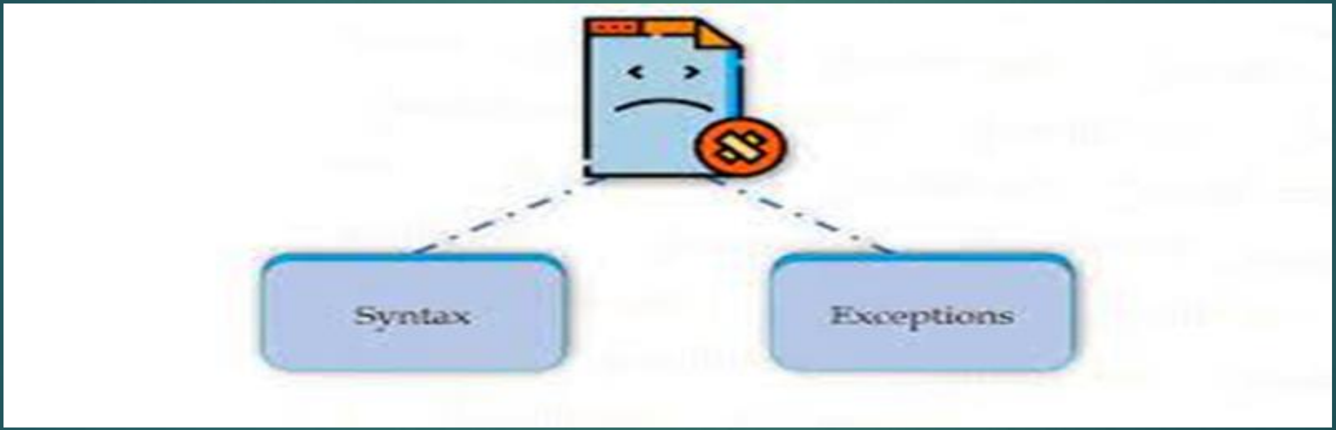
errors that happens during the execution of a program that could be handled

Errors in Programming

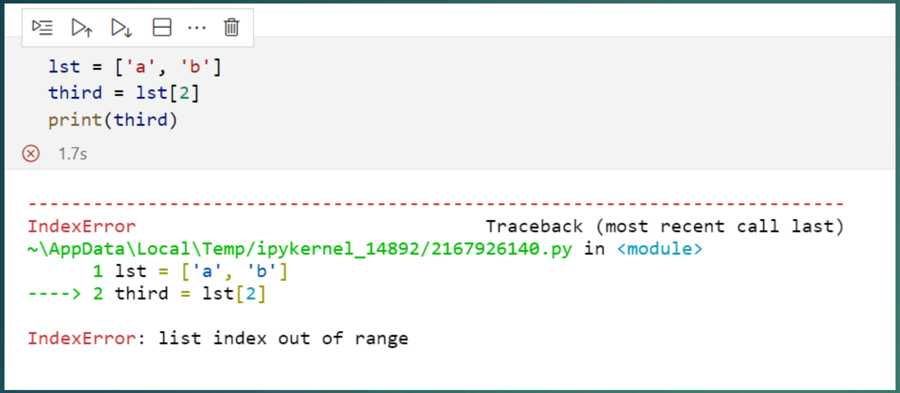
Errors are the problem or fault occurs in the program that make the behavior of the program abnormal

We consider two types of errors mainly

1. **Syntax Errors** : *mistake in coding structure, can't ignore or handled*
2. **Exceptions** : *during the execution, can be handled.*



Handling an Exception: Flow of control

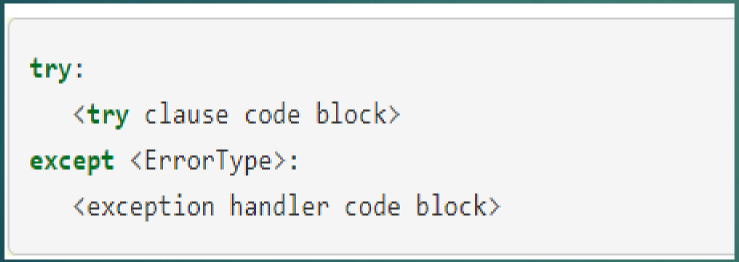


```
lst = ['a', 'b']
third = lst[2]
print(third)
```

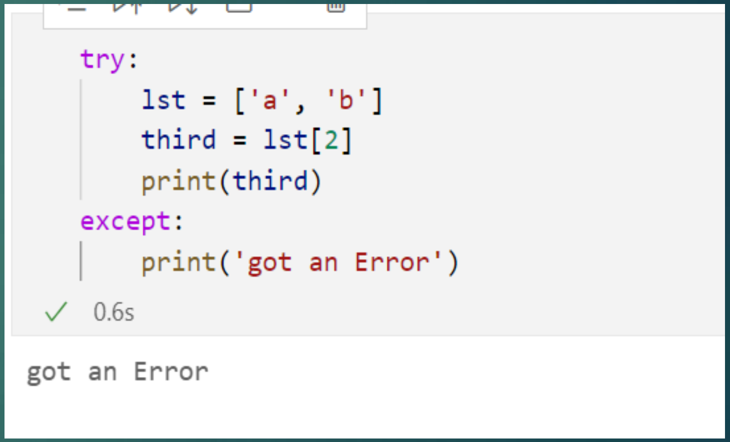
```
1.7s
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
~\AppData\Local\Temp/ipykernel_14892/2167926140.py in <module>
      1 lst = ['a', 'b']
----> 2 third = lst[2]

IndexError: list index out of range
```

1. try-except



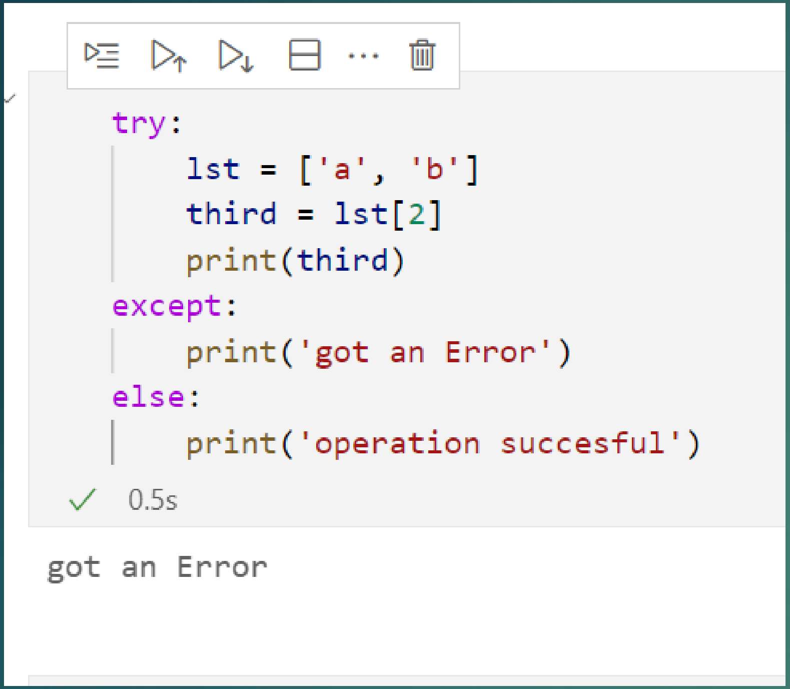
```
try:
    <try clause code block>
except <ErrorType>:
    <exception handler code block>
```



```
try:
    lst = ['a', 'b']
    third = lst[2]
    print(third)
except:
    print('got an Error')
```

```
0.6s
got an Error
```

2. try-except-else

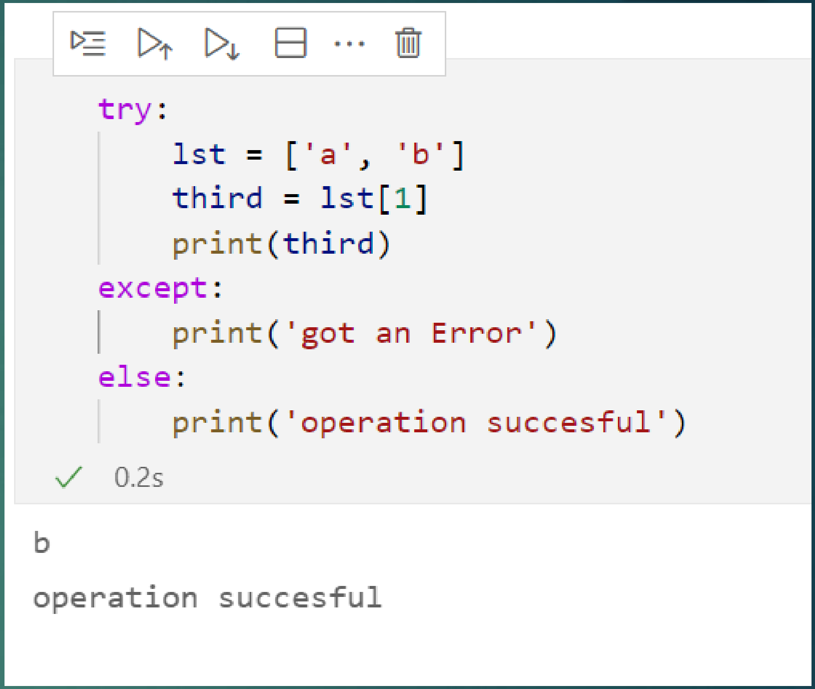


```
try:
    lst = ['a', 'b']
    third = lst[2]
    print(third)
except:
    print('got an Error')
else:
    print('operation succesful')
```

✓ 0.5s

```
got an Error
```

With Exception



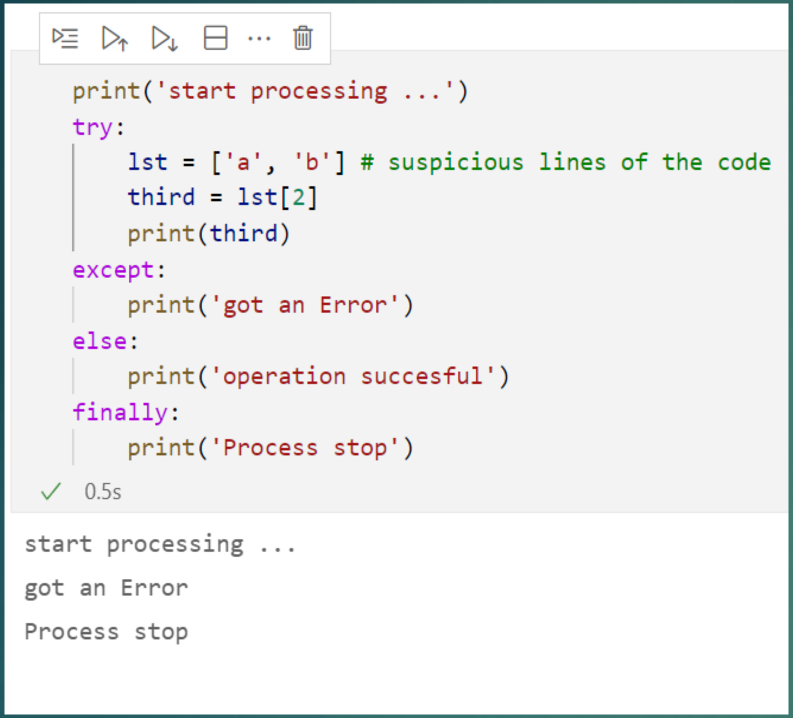
```
try:
    lst = ['a', 'b']
    third = lst[1]
    print(third)
except:
    print('got an Error')
else:
    print('operation succesful')
```

✓ 0.2s

```
b
operation succesful
```

No Exception

3. try-except-else-finally or try-except-finally

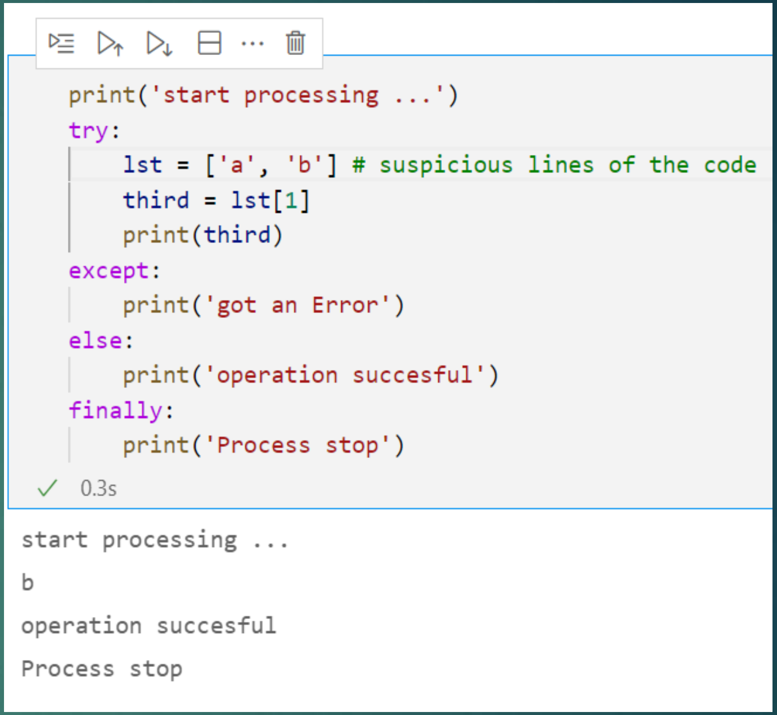


```
print('start processing ...')
try:
    lst = ['a', 'b'] # suspicious lines of the code
    third = lst[2]
    print(third)
except:
    print('got an Error')
else:
    print('operation succesful')
finally:
    print('Process stop')
```

✓ 0.5s

```
start processing ...
got an Error
Process stop
```

With Exception



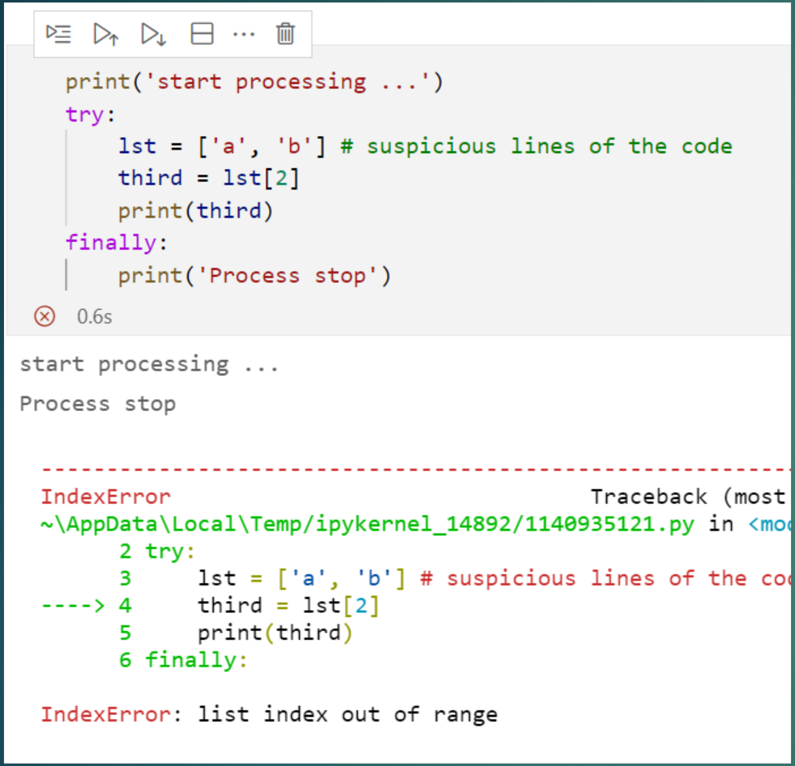
```
print('start processing ...')
try:
    lst = ['a', 'b'] # suspicious lines of the code
    third = lst[1]
    print(third)
except:
    print('got an Error')
else:
    print('operation succesful')
finally:
    print('Process stop')
```

✓ 0.3s

```
start processing ...
b
operation succesful
Process stop
```

No Exception

4. try-finally: In this combination we don't handle the exceptions but important lines we can run.



```
print('start processing ...')
try:
    lst = ['a', 'b'] # suspicious lines of the code
    third = lst[2]
    print(third)
finally:
    print('Process stop')
```

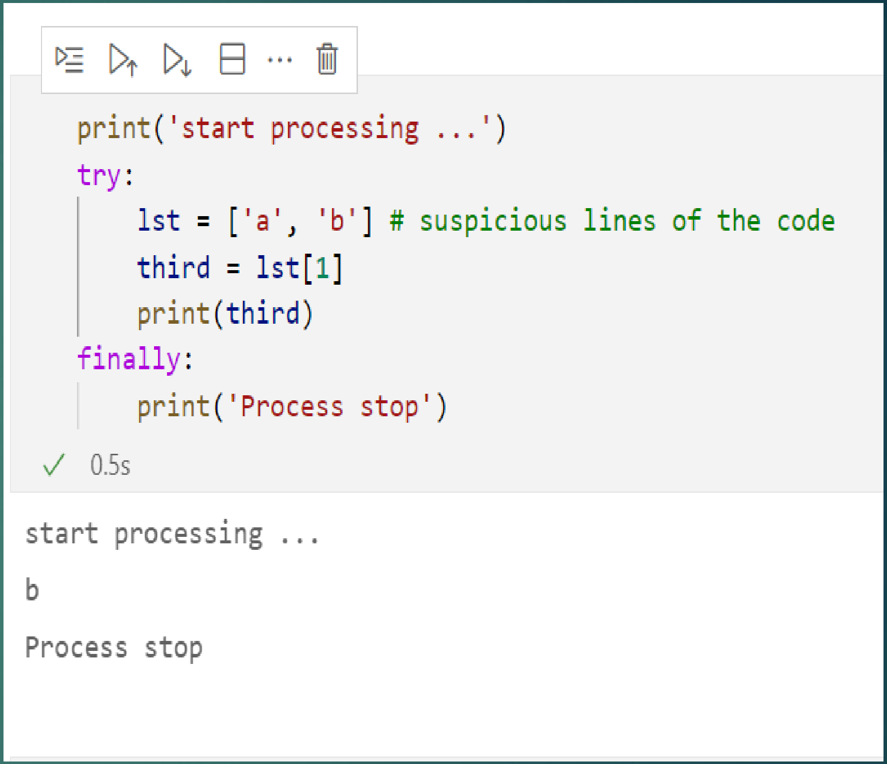
0.6s

```
start processing ...
Process stop

---------------------------------------------------------------------------
IndexError                                Traceback (most
~\AppData\Local\Temp/ipykernel_14892/1140935121.py in <mod
      2 try:
      3     lst = ['a', 'b'] # suspicious lines of the co
----> 4     third = lst[2]
      5     print(third)
      6 finally:

IndexError: list index out of range
```

With Exception



```
print('start processing ...')
try:
    lst = ['a', 'b'] # suspicious lines of the code
    third = lst[1]
    print(third)
finally:
    print('Process stop')
```

✓ 0.5s

```
start processing ...
b
Process stop
```

No Exception

Standard Exception in Python3

Here is a list all the standard Exceptions available in Python –

- Exception

Base class for all exceptions

- ZeroDivisionError

Raised when division or modulo by zero takes place for all numeric types.

- 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.

- StopIteration

Raised when the next() method of an iterator does not point to any object.

- ArithmeticError

Base class for all errors that occur for numeric calculation.

Continue…

- 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.
- 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

- 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.

Multiple except blocks

```
try:
    statement(s)
except Exception1:
    statement
except Exception2:
    statement
except Exception3:
    statement
```

Multiple Exceptions in a Single except block

```
try:
    n=int(input('Enter the number'))
    print(n**2)
except (KeyboardInterrupt, ValueError,TypeError):
    print("Please Check before you enter")
print("Bye")
```

1. ZeroDivisionError

```
 a=int(input('Enter the number'))
 b=int(input('Enter the number'))
try:
    c=a/b
      print(c)
except ZeroDivisionError as e:
     print("Please Check Denominator ",e)
print("Bye")
```

2. ValueError

```
try:
    n=int(input('Enter the number'))
    print(n**2)
except ValueError as e:
     print("Please Check before you enter ",e)
print("Bye")
```

3. KeyboardInterrupt

```
try:
    n=int(input('Enter the number'))
    print(n**2)
except KeyboardInterrupt as e:
     print("Do not press ctrl+c ",e)
print("Bye")
```

4. TypeError

```
try:
    n=int(input(‘Enter the number’))
    print(n**2+’Hello’)
except TypeError as e:
     print(“Please Check the datatype in
     expression”,e)
print(“Bye”)
```

5. FileNotFoundError

```
fname=input("Enter the filename")
try:
    fp=open(fname,'r')
    print(fp.read())
except FileNotFoundError as e:
     print("Please Check before you enter ",e)
print("Bye")
```

6. IndexError

```
L=[1,2,3,4]
try:
    print(L[9])
except IndexError as e:
    print(“Please Check the index ”,e)
print(“Bye”)
```

7.NameError

```
try:
    print(a)
except NameError as e:
     print(“Please Check the variable a ”,e)
print(“Bye”)
```

8. KeyError

```
D={1:1,2:8,3:27,4:64}
try:
    print(D[5])
except KeyError as e:
    print("Please Check the key exists or not ",e)
print("Bye")
```

Raising Exceptions

- We can deliberately raise an exception using the raise keyword
- Syntax

raise [Exception [args]]

Example:

```
try:
    n=10
    print(n)
    raise ValueError
except:
      print("Exception Occurred")
```

Re-raise an Exception

```
try:
    raise NameError
except:
    print(“Re-raising the exception”)
    raise
```

Review..

- 1. Why need Exception Handling?
- 2. What is the process of Exception Handling?
- 3. Explain the use of else clause in exception handling.
- 4. when we use the finally block?
- 5. Explain NameError, KeyError, ValueError and ZeroDivisionError.
- 6. Base class for all Exceptions?

Programs

1. WAP that prompts the user to enter a number. If the number is positive or zero, print it, otherwise raise an exception ValueError with some message.
2. WAP which infinitely print natural numbers. Raise the StopIteration exception after displaying first 20 numbers to exit from the program.
3. Compute the reciprocal for all elements in a given list and handle exceptions if found.

   lst = [1, 3, 0, ‘A’, 34, 10]

Tricky Questions (Code 1)

1. What will be the output?

```
def fun():
    try:
        print("Hello")
        return 1
    finally:
        print("DONE")
fun()
print("Hi")
```

2. What will be the output?

```
def fun():
    for i in range(5):
        try:
            if i==0:
                print("Hello")
                break
        finally:
            print("DONE")
fun()
print("Hi")
```

3. What will be the output?

```
def fun():
    for i in range(5):
        try:
            if i==0:
                print("Hello")
                continue
        finally:
            print("DONE")

fun()
print("Hi")
```