Queen Mary
University of London

QHP4701
Introduction to Data Science Programming

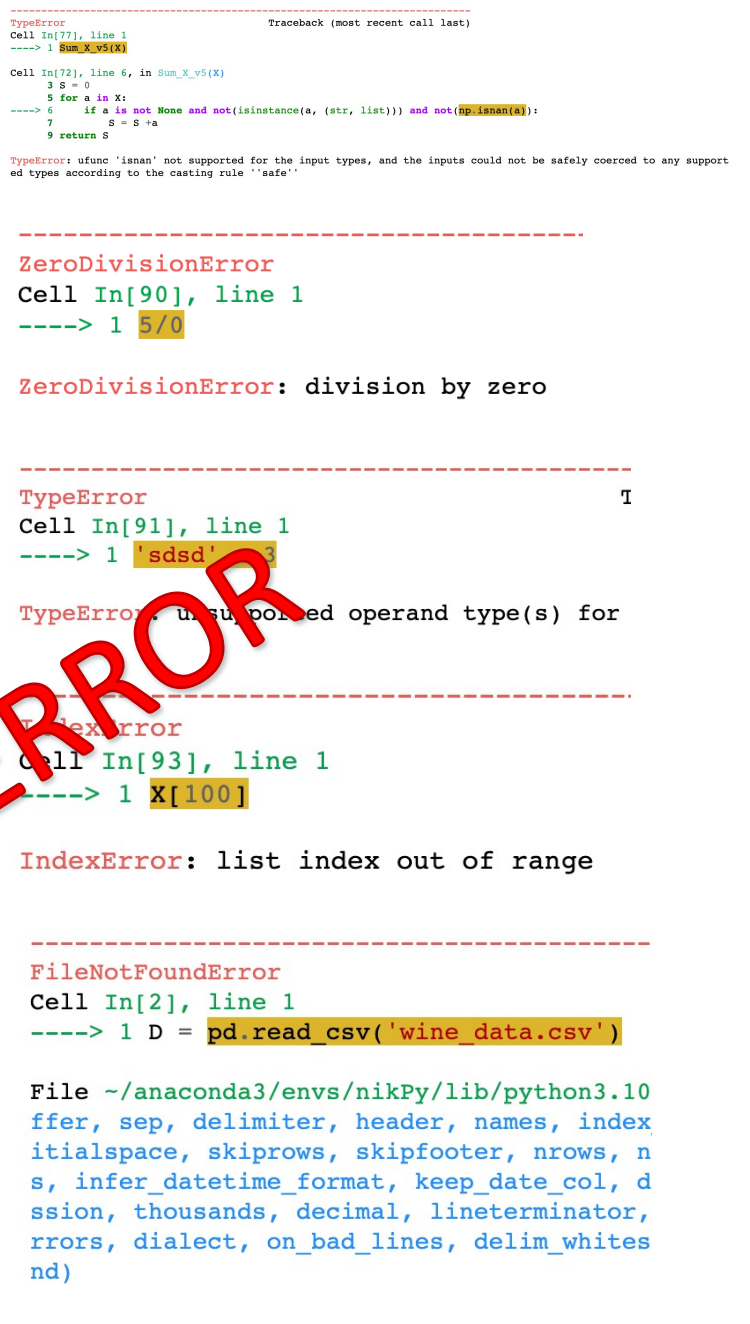
Error Handling

Lecturer: Nikesh Bajaj, PhD
School of Physical and Chemical Sciences
http://nikeshbajaj.in

Lecture Outline

Error Handling in Python

- Error Handling
- Common Errors
- Efficient ways: using conditions
- Try-Except routine
- Assert/Raise Keywords
- Documentation: docstring
- Analyse code with print: Debugging
- Next: Error full Notebook



Error Handling

Errors: when you don’t get what you want.

- First step to avoid errors, *DO NOT GET ANY ERROR!*
- **Read the Error Message VERY carefully** to understand the issue.
- A computer program has be very specific, with explicit instructions, so that it knows what to do in every conditions. *BE SPECIFIC.*
- While writing a code, make some test cases to test your code. (software testing)
- Write a good docstring for others to understand the expected input and output.

Lecture Outline

Error Handling in Python

- Error Handling
- Common Errors
- Efficient ways: using conditions
- Try-Except routine
- Assert/Raise Keywords
- Documentation: docstring
- Analyse code with print: Debugging
- Next: Error full Notebook

Common Errors

You are familiar of most of the errors by now, while working of lab sheets and you are also familiar how to avoid them

- IndexError — list index out of range / out of bounds
- ValueError — could not convert
- TypeError — unsupported operand type(s)
- KeyError — key not in dictionary
- ZeroDivisionError — division by zero
- FileNotFoundError: — [Errno 2] No such file or directory:
- ModuleNotFoundError: — No module named 'pnadas'
- AttributeError — wrong method or attribute name

Read the Error Message VERY carefully

It explains everything!

Lecture Outline

Error Handling in Python

- Error Handling
- Common Errors
- Efficient ways: using conditions
- Try-Except routine
- Assert/Raise Keywords
- Documentation: docstring
- Analyse code with print: Debugging
- Next: Error full Notebook

Efficient ways : using conditions

Write a program to compute sum of all the numbers in a list or an array X

- X = [1, 2, 3, 0, 1, 4]
- X = [1,2,3, None, 0, 4, None, 2]
- X = [1,2,3, None, 0, 4, None, 2, 'A']
- X = [1,2,3, None, 0, 4, None, 2, np.nan]
- X = [1,2,3, None, 0, 4, None, 2, 'A' , np.nan]
- X = [1,2,3, None, 0, 4, None, 2, 'A' , np.nan, [1, 2]]
- X = [1,2,3, None, 0, 4, None, 2, 'A' , np.nan, [1], {1,2}, {'a':1}]
- X = [1,2,3, None, 0, 4, None, 2, 'A' , np.nan, [1], {1,2}, {'a':1}, '1.2']

Efficient ways : using conditions

Sum of all the numbers in a list or an array X

- Writing in an efficient way to avoid errors

```
def Sum_X(X):
    S = 0
    for a in S:
        if not cond: #some conditions
            S = S + a
    return S
```

Efficient ways : using conditions

Sum of all the numbers in a list or an array X

- Multiple conditions

```
def Sum_X(X):
    S = 0
    for a in S:
        if cond1 and cond2: #some conditions
            S = S + a
    return S
```

- Python is lazy in testing the conditions,. It always test first condition then second.

Efficient ways : using conditions

Exclusion Criteria vs Inclusion Criteria

- Try using small set of conditions.
- Instead of all the conditions to exclude, a fewer condition in include can be used.

```
def Sum_X_v1(X):
    S = 0
    for a in S:
       if not cond1 and not cond2 and not cond3:
          S = S + a
    return S
```

```
def Sum_X_v2(X):
    S = 0
    for a in S:
       if condA and condB:
          S = S + a
    return S
```

Lecture Outline

Error Handling in Python

- Error Handling
- Common Errors
- Efficient ways: using conditions
- Try-Except routine
- Assert/Raise Keywords
- Documentation: docstring
- Analyse code with print: Debugging
- Next: Error full Notebook

Try-Except routine

In Python *try-except* routine allows code to continue without stopping in case of any error. except section here allows you to handle the errors

```
try:
    S = 0
    #some code that may or may not throw an Error
except:
    print('Something is wrong!!')
    #If ANY error occurs, this section of code is executed
    #define here, how you like to handle the error
```

- It is a way to **try** running a code, else do something else.

Try-Except routine

While using *try-except* routine, we can print error messages, while continuing the code. It is called catching an error.

```
try:
    S = 0
    #some code that may or may not throw an Error
except Exception as e:
    print('Something is wrong!!')
    print(type(e))  #Type of Error
    print(e)        #Error Message
```

Exception
|— MemoryError
|— NameError
|— OSError
|— ReferenceError
|— RuntimeError
|— StopAsyncIteration
|— StopIteration
|— SyntaxError
|— SystemError
|— TypeError
|— ValueError

Check the full list of Exceptions and details: https://docs.python.org/3/library/exceptions.html

Try-Except routine

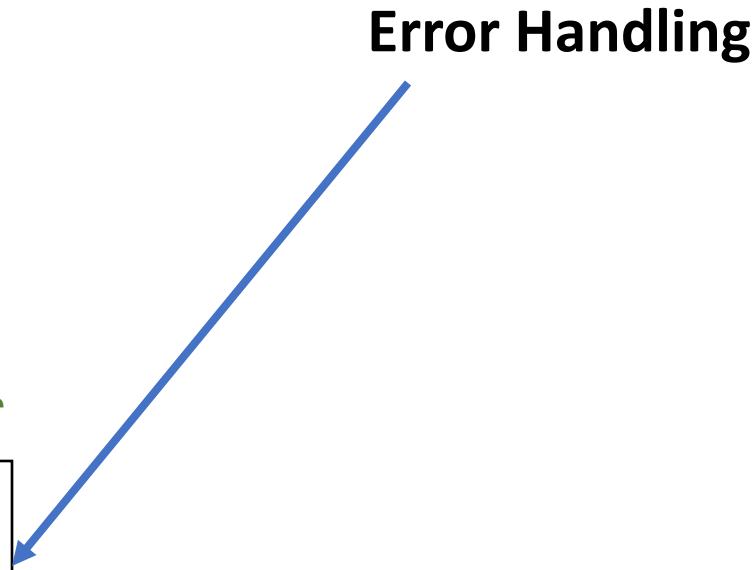
Any specific type of exception can be caught with custom message

```
try:
    S = S + a
    #some code that may or may not throw an Error
except NameError:
    print('Either S or a is not defined')
except:
    print('Something else is wrong!!')
```

Exception
|— MemoryError
|— NameError
|— OSError
|— ReferenceError
|— RuntimeError
|— StopAsyncIteration
|— StopIteration
|— SyntaxError
|— SystemError
|— TypeError
|— ValueError

Try-Except routine

A full structure of try-except routine




```
try:
    S = S + a
    #some code that may or may not throw an Error
except NameError:
    print('Either S or a is not defined')
except:
    print('Something else is wrong!!')
else:
   print('HURREY!! NO ERRORS') # If NO Errors
finally:
   print('We always do run')   # This code is always executed
```

Lecture Outline

Error Handling in Python

- Error Handling
- Common Errors
- Efficient ways: using conditions
- Try-Except routine
- Assert/Raise Keywords
- Documentation: docstring
- Analyse code with print: Debugging
- Next: Error full Notebook

Assert/Raise Keywords

Custom Error Messages and conditions

- There are two ways to throw an Error and stop the execution of code if certain conditions are met.
- assert, is used to make sure some conditions are met before proceeding to code.

```
def sum_squqre(x,y):
    assert x!=0
    # x cannot be zero

    z = (x**2 + y**2)/x
    return z
```

$$z = \frac{x^2 + y^2}{x}$$

Assert/Raise Keywords

Custom Error Messages and conditions

- raise, is used to stop the code by throwing an error (raising an error) with a message.

$$c = \sqrt{x^2 + y^2} + \sqrt{z}$$

```
def sum_squqre(x,y,z):
    if z<0:
       raise ValueError('z can not be negative')
    c1 = (x**2 + y**2)**(1/2)
    c2 = (z)**(1/2)
    c = c1 + c2
    return c
```

- Mathematically valid but c becomes a complex number for a negative z.

Lecture Outline

Error Handling in Python

- Error Handling
- Common Errors
- Efficient ways: using conditions
- Try-Except routine
- Assert/Raise Keywords
- Documentation: docstring
- Analyse code with print: Debugging
- Next: Error full Notebook

Documentation with docstring

Do write docstring

explain what is expected

as input.

$$c = \sqrt{x^2 + y^2} + \sqrt{z}$$

```
def sum_squqre(x,y,z):
    '''
    This function computes c = sqrt(x**2 + y**2) + sqrt(z)
    Input:
    x: a real value
    y: a real value
    z: a positive real value z>=0

    Output:
    c: a real value
    '''
    if z<0:
       raise ValueError('z can not be negative')
    c1 = (x**2 + y**2)**(1/2)
    c2 = (z)**(1/2)
    c = c1 + c2
    return c
```

Lecture Outline

Error Handling in Python

- Error Handling
- Common Errors
- Efficient ways: using conditions
- Try-Except routine
- Assert/Raise Keywords
- Documentation: docstring
- Analyse code with print: Debugging
- Next: Error full Notebook

Analyse code using print: Debugging

While developing a code, it is common to see unexpected results or error.

- Using print, is an easy way to trace what is happing in the code. Using print at every stage to display results to see when and where Error occurs.

```
def Freq_Char(S):
    freq = {}
    for c in S:
        if cond:
            freq[c] = freq[c] +1
    return freq
```

```
def Freq_Char(S):
    freq = {}
    print(S)
    for c in S:
        print('char', c)
        if cond:
            freq[c] = freq[c] +1
        print(freq)
    return freq
```

- Once code is working as expected, print lines can be removed

Lecture Outline

Error Handling in Python

- Error Handling
- Common Errors
- Efficient ways: using conditions
- Try-Except routine
- Assert/Raise Keywords
- Documentation: docstring
- Analyse code with print: Debugging
- Next: Error full Notebook

- Next !!!
  - 4.4: Lab session on Error Handling

    You will be given a notebook full of errors and your job will be to fix those error

Queen Mary

University of London