Dynamic Modelling Course - TEP4290: Warm-up 13

The point of this exercise is for you to get familiar with the main errors that you will encounter while programming in Python and how to solve them.

Bascally, there exists different types of errors which will hint you to where to look for and debug (fix) it. By learning to do this, you should learn to become a more independent programmer and translate similar issues that other people have had into your own context.

We will learn about SyntaxError, KeyError, ValueError amongst other.

Useful resources include:

- https://aguaclara.github.io/aguaclara_tutorial/python/python-common-errors.html
- https://edcarp.github.io/2018-11-06-edinburgh-igmm-python/07-errors/index.html
- https://learningactors.com/python-keyerror-exceptions-and-how-to-handle-them/

For catching exceptions:

- https://www.tutorialsteacher.com/python/exception-handling-in-python

About debugging with a standard python package:

- https://www.youtube.com/watch?v=ChuU3NlYRLQ

The exercise is pass/fail.

Good luck!

Errors and Exceptions

No matter your skill as a programmer, you will eventually make a coding mistake. Such mistakes come in three basic flavors:

- *Syntax errors:* Errors where the code is not valid Python (generally easy to fix)
- *Runtime errors:* Errors where syntactically valid code fails to execute, perhaps due to invalid user input (sometimes easy to fix)
- *Semantic errors:* Errors in logic: code executes without a problem, but the result is not what you expect (often very difficult to track-down and fix)

Here we're going to focus on how to deal cleanly with *runtime errors*. As we'll see, Python handles runtime errors via its *exception handling* framework.

Debugging

While there is a whole world of debugging culture out there - often with add-ons or dedicated programs for debugging, but you don't have to go so far to effectively debug. We already spoke about syntax and runtime errors, but you will likely spend a lot of time on figuring out semantic errors or runtime errors connected to semantic errors.

The easiest way of debugging is using the __print__ statement!

While you may use any workflow that pleases you, just inserting print statements in your code to check what your variables do is often more than enough to understand the mistage you get.

Imports

first we import what we need to for this assignment:

In [36]:
```
import numpy as np
import pandas as pd
```

Tasks

By looking at the errors that are returned, fix the following Runtime Errors

If you've done any coding in Python, you've likely come across runtime errors. They can happen in a lot of ways.

For example, if you try to reference an undefined variable:

In [37]:
```
Q = 6
print(Q)
```

```
6
```

Or if you try an operation that's not defined:

In [38]:
```
1 + 2
'abc' + 'def'
```

Out[38]:
```
'abcdef'
```

Or you might be trying to compute a mathematically ill-defined result:

In [39]:
```
2 / 1
```

Out[39]:
```
2.0
```

Or maybe you're trying to access a sequence element that doesn't exist:

In [40]:
```
L = [1, 2, 3]
L[1]
```

Out[40]:
```
2
```

Let's fix some Syntax Errors too

PS: Syntax error, much like in a normal sentence, often means you typed something wrong.

Useful: https://www.raiseupwa.com/miscellaneous/what-is-syntax-error-in-python-example/

In [41]:
```
x = 0

while x < 5:
    print('x is smaller than 5')
    x +=1
```

```
x is smaller than 5
x is smaller than 5
x is smaller than 5
x is smaller than 5
x is smaller than 5
```

In [42]:
```
for x in range(5):
    y = x + 1
```

Attribute errors are also common when starting to program. Can you fix these?

In [43]:
```
my_tuple = [1, 2, 3]
my_tuple.reverse()
```

In [44]:
```
data = pd.DataFrame({'Data': [1,2,3,4]})
```

In [45]:
```
# A little refresher for syntax errors
data.head()
```

Out[45]:

	Data
0	1
1	2
2	3
3	4

Type errors:

In [46]:
```
def sum(x, y):
      z = x + y
      return z

sum(3, 4)/2
```

Out[46]: 3.5

In [47]:
```
x = [1,2,3]
y = [4,5,6]

x + y
```

Out[47]: [1, 2, 3, 4, 5, 6]

Finally, Key Errors

In [48]:
```
data = pd.DataFrame({'Year': [2000, 2001, 2002], 'Inflow': [100, 120, 150]})

# Find the value for year 2001
data.loc[data['Year']==2001, 'Inflow'].values
```

Out[48]: array([120])

Finding errors with print statements

The next cell contains a simple construct of two functions and will result in an error if you use 102 as the input.

Your task is to find what the error is, and where it occurs - what are the variables related to the error, what is their state? For this purpose, use print() within the function to get the exact values of everything that might be relevant!

You do not need to fix the error!

you can try to change the input for the last function to a different seed - what is the changed state now?

In [49]:
```python
def do_random_shit(seed:int):
    '''Just does something weird and unesseary.'''
    #set seed to make sure we always get the same
    np.random.seed(seed)

    #create two random arrays of length 20 and scale them up
    array1 = np.random.rand(20)*100
    array2 = np.random.rand(20)*100
    print('Array2:', array2)

    results = np.subtract(array1, array2)
    print('Results:', results)

    new_results = random_scaling(results)
    return new_results


def random_scaling(array):
    '''Simple scaling of a random array with the input array .'''
    array1 = np.random.rand(np.size(array))*100 #random array that will be scaled
    print('random scaling array1:', array1)
    scaled_results = [] #emtpy result list
    for number, base in zip(array, array1): #iterating through both arrays simultan
        rounded_number = int(number) #rounding the scaling factor to an int
        print('Rounded number:', rounded_number)
        rounded_base = int(base)
        scaled_number = rounded_base/rounded_number
        scaled_results.append(scaled_number)

    return scaled_results


do_random_shit(105)
```

```
Array2: [53.61085252 17.31619728 16.59375783 11.04694724 32.58528804 89.74140359
 54.84370422 24.13703223 32.80036008 58.89284048 79.51706046 48.11718456
 57.61860106 59.01139873 79.78978019 77.8123727  66.66474271 51.12254906
 53.63547872 17.32220015]
Results: [-45.23686013  16.03443197  69.34181668  -0.47826371  20.97871637
 -83.37244662 -25.16024635 -23.91250855  12.31123086  29.46912015
 -21.81130043 -23.22941268 -23.46329565 -51.99157011 -25.94225499
 -25.26399989  15.21492098  17.51050929   9.46123348  19.71671319]
random scaling array1: [67.57450159 90.81579981 53.5487007  18.42208873  9.1359579
57.10584311
 82.4650321  15.72205997 65.86791614  9.19361201 94.84933192  9.57753628
 42.94092622 62.6999762  77.04111163 47.32475018 27.96763837 24.83980175
  9.77562712 75.45562096]
Rounded number: -45
Rounded number: 16
Rounded number: 69
Rounded number: 0
```

```
---------------------------------------------------------------------------
ZeroDivisionError                         Traceback (most recent call last)
Cell In[49], line 33
     28         scaled_results.append(scaled_number)
     30     return scaled_results
---> 33 do_random_shit(105)

Cell In[49], line 14, in do_random_shit(seed)
     11 results = np.subtract(array1, array2)
     12 print('Results:', results)
---> 14 new_results = random_scaling(results)
     15 return new_results

Cell In[49], line 27, in random_scaling(array)
     25     print('Rounded number:', rounded_number)
     26     rounded_base = int(base)
---> 27     scaled_number = rounded_base/rounded_number
     28     scaled_results.append(scaled_number)
     30 return scaled_results

ZeroDivisionError: division by zero
```

In [ ]:

answering the last question

the state of the different variables and what caused the error:

error: it is a mathmatical error that some of denominator becomes 0

variables:

results list: this list contains all the values which will be treated to be denominator in the iteration
rounded_number: this variable is a rounded number of values from results list, and it is denominator for next calcuation