PYTHON COURSE

Python course , from scratch to profissionalism ...

Made by : Taha Khaldoun Amro

دورة تعليم لغة بايثون

Errors , Exceptions Raising

أحد الأمور التي تتميز بها لغة بايثون هي سهولة فهم الerrors التي تعرض للمستخدم أثناء تنفيذ الكود , حيث أن بعض الerrors التي تطبع للمستخدم تبدو و كأنها لغة انجليزية عادية ,ذلك على عكس لغات البرمجة الأخرى التي تكون الerrors فيها معقدة و غير مفهومة الا للمتمرسين في اللغة .

يعبر مفهوم الexceptions عن الرسالة التي تطبع للمستخدم في الterminal لتصف له خطأً في الكود , و هناك أنواع كثيرة للأخطاء أهمها : (KeyError , IndexError , SyntaxError ...) و تعمل لغة البرمجة بايثون على تعقب موقع الerror داخل الكود من خلال مفهوم الTracebacking و هذا ما يميز هذه اللغة عن الكثير من لغات البرمجة .

لنقل أنك بنيت تطبيقاً يعمل على حساب معادلة معينة , و اتى شخص ليستخدم تطبيقك , قام المستخدم الذي لا يعلم كيفية استخدام تطبيقك بادخال قيم نصية بدلاً من قيم رقمية , ستظهر له بطبيعة الحال الError التلقائي للغة , قد يكون هذا الشخص غير عالم باللغة , لذلك لن يفهم الerror , و قد يعتبره مشكلة في كودك , و لكن لا تقلق , فقد وفرت لنا لغة بايثون امكانية تنسيق errors خاصة بنا , تكون مفهومة بشكل أكبر للمستخدم .
قد يتساءل البعض , ما هو الفرق بين الerrors التي نقوم نحن بتصميمها و الرسائل العادية التي نطبعها للمستخدم في حال حدوث خطأ , تكمن الاجابة في أن الرسائل العادية لا توقف عمل الكود , و لكن الerror raising يفعل .

Errors , Exceptions Raising

سنتعلم مفهوم الExceptions raising من خلال نشاط عملي يمكننا من فهم الأمر بشكل مفصل .
سنعمل الان على بناء تطبيق بسيط , قم بتعريف متغير له قيمة عددية , قم بعمل نظام يفحص اذا ما كان العدد اقل من صفر , و اجعله يطبع رسالة تقول (the number x is less than 0) اذا كان العدد اصغر من صفر , اما اذا كان العدد أكبر من 0 او يساويه فاطبع الرسالة (the number x is good) حيث x هي العدد , بعد تنفيذ الجمل الشرطية , اطبع الرسالة (the code is done) .
هل لاحظت شيءا ؟ نعم لا يوجد أي أخطاء في الكود لذلك لا يتم رفع أي exception و هذا ما سنعمل عليه الان .
في تطبيقك الذي بنيته , تريد اخبار المستخدم بأنه يجب ادخال عدد أكبر أو يساوي صفر , لذا فانك بحاجة الى رسالة توقف الكود عن العمل لأن ما تم ادخاله خطأ , حيث أن البرنامج يقبل الأعداد التي هي أكبر او تساوي صفر فقط , و لكن كيف تتم العملية ؟
توفر لنا اللغة أمر ارجاع خاص بالerrors المصممة من قبل المستخدم , يعمل الأمر بنفس مبدأ عمل امر الreturn و يمكننا القول بأنه خاص بهذه العملية , فهو لا يعمل فقط على ارجاع error , بل يعمل أيضا على ايقاف سير الكود .

Errors , Exceptions Raising

```
x = int(input())

if x < 0:
  raise Exception('cant have a number less than 0')

else :
  print('Good number .')

print('Code is done (: )')
```

لا بد في النهاية أنك حصلت على كود بهذا الشكل , جرب الان أن تقوم بادخال قيمة أكبر من 0 تارة , و تارة أخرى أدخل رقمأ اقل من 0 و انظر ماذا سيحدث .
ستلاحظ في حال تنفيذك للكود انك لو أدخلت عددا اقل من 0 سيتم ايقاف الكود و لن تطبع الرسالة في النهاية , و هذا ما نبحث عنه .

Errors , Exceptions Raising

اذا فان الexceptions توقف عمل الكود تماما , و نلاحظ في الرسالة التي كتبت لنا أنه قد ذكر لنا موقع الError في الكود أو في هذه الحالة موقع أمر رفع الخطأ للمستخدم (exception code line) .
و لكن اذا فكرنا بالأمر بشكل منطقي , الا يبدو أن الerror الذي يحصل هنا هو ال ValueError الموجود في لغة بايثون ؟
حيث أن هذا الخطأ يظهر عند عدم تناسق أنواع البيانات , حيث تمكننا لغة بايثون من تحديد نوع الexception الذي نرفعه للمستخدم :

```
x = int(input())

if x < 0:
  raise ValueError('cant have a number less than 0')

else :
  print('Good number .')

print('Code is done (:)')
```

Exceptions Handling

تعلمنا الان كيفية استخدام الexceptions و رفعها للمستخدم , و لكن لنقل مثلا أنك تريد بناء روبوت ما يقوم بمهام متعددة , في حال حدوث أي خطب ما في أي من قطع الروبوت فان الكود الموجود داخله يعمل على رفع error و يتوقف الكود , مما يوقف الروبوت , أنت لا تريد للروبوت أن يتوقف , تريده أن يخبرك بموقع المشكلة و أن يتابع العمل بشكل طبيعي .

هنا يكمن مفهوم الexceptions handling و الذي يعني التعامل مع الerror قبل حدوثها , و هو شكل من أشكال الجمل الشرطية الخاصة بموضوع الerrors و الذي يعمل بمجموعة من الأوامر على النحو الاتي :

1. try :

يوضع الأمر try قبل الكود كله , حيث أنه يقوم بتفعيل الكود بشكل طبيعي , و لكن الفرق هنا أنه ينتظر حدوث error للتعامل معه .

2. except :

لا يمكننا استخدام هذا الأمر من دون استخدام الأمر السابق قبله , و يكون الكود تحت هذا الأمر هو ما سيتم تنفيذه في حال حدوث error معين , فهو مثل الجملة الشرطية ,يعمل كالتالي : في حال حدوث error اثناء تجريبك للكود , نفذ الكود التالي .

Errors , Exceptions Raising

سنحاول الان بناء نظام بسيط يعمل على حساب عمر الشخص من خلال ادخاله لعام ميلاده فقط .
قم بكتابة كود يعمل على ادخال رقم من المستخدم و من ثم طباعة عمره من خلال طرحه من العام الحالي (2023) على سبيل المثال :

```
year_of_bearth = int(input())

print(f'Your age is :{2023 - year_of_bearth}')
```

لو فكرنا في الموضوع بشكل منطقي بعض الشيء , ماذا لو أدخل المستخدم كلمة او أي شيء غير الرقم ؟
ببساطة فان الكود سيظهر لنا ValueError مفاده أنه لا يمكننا طرح نص من رقم و لهذا السبب سنستخدم الtry :

```
try :
  year_of_bearth = int(input())
  print(f'Your age is :{2023 - year_of_bearth}')
except ValueError:
  print('please insert a number')
```

Exceptions Handling

ما يحدث في الكود السابق ببساطة هو أن النظام يعمل على تنفيذ الكود , في حال وجود الValueError لأي سبب من الأسباب , فان النظام يوقف العملية و يطبع : please insert a number و هذا هو كل ما يتمحور حوله الأمر .

من الجدير بالذكر أن العملية except يمكن استخدامها دون تحديد نوع الerror الذي سيتم التعامل معه , ولكن ما يحدث هنا ببساطة هو أن الكود تحت عملية الexcept سيتم تنفيذه في أي error سيحدث مهما كان نوعه .

جرب بنفسك الان القيام بتجربة بسيطة , لو جربت طباعة عدد مقسوم على 0 سيظهر لك ZeroDivisionError , و ذلك لان المنطق الرياضي يمنع ذلك , جرب بنفسك كتابة كود يطلب قيمتين من المستخدم , الأولى تكون هي العدد المقسوم و الثانية هي المقسوم عليه , باستخدام الExceptions handling اكتب كوداً يطبع الرسالة : cant divide on zero اذا ما تم ادخال 0 في المقسوم عليه .

هذا كان درسنا لليوم , نترك لكم الرابط التالي لمزيد من الأمثلة و الشرح :
https://www.youtube.com/watch?v=LBf_8txij3I&list=PLDoPjvoNmBAyE_gei5d18qkfIe-Z8mocs&index=91&pp=iAQB