Exception Handling Part 01

Before we start, we will elaborate on exception handling as much as possible. Be sure to distinguish between **exceptions** and **errors**

What is an error? What is an exception? ¶

First, let’s take a look at what is an error

Error is also called analysis error, which means that the belt does not meet the specifications and cannot be run at all.

In [4]:

```
# demo
while True print('Hello world')
```

```
  File "<ipython-input-4-42ea4a5d202b>", line 2
    while True print('Hello world')
                   ^
SyntaxError: invalid syntax
```

The parser outputs the line with the syntax error and displays an "arrow" pointing to the first error detected in this line. The error is caused by the token above the position indicated by the arrow (or at least it was detected here): In the example, the error was detected in the print () function because there is no colon (':') in front of it . The file name and line number are also output so that you can know where to check when the input comes from the script file.

Next, let’s take a look at what is an exception.

Even if the code is expressed correctly, it may still cause problems at runtime. Just like "I like to eat TV." this sentence has no grammar mistake, however, it doesn't make sense. Since TV is not food.

In [1]:

```
# demo
b=123/0
```

```
---------------------------------------------------------------------------
ZeroDivisionError                         Traceback (most recent call last)
<ipython-input-1-5f59c11e5a86> in <module>
      1 # demo
----> 2 b=123/0

ZeroDivisionError: division by zero
```

The code above causes **ZeroDivisionError**, an exception. There is no error in the expression of b = 123/0, but there must be a problem when running, this is the exception
For various information about built-in exceptions, we can refer to: https://docs.python.org/zh-cn/3/library/exceptions.html#bltin-exceptions (https://docs.python.org/zh-cn/3/library/exceptions.html#bltin-exceptions)

Exception Handling

Python provides methods that can handle exceptions. I will provide an example that requires the user to keep typing until a valid integer is entered (of cause the user can interrupt the operation just simply use control+c).

In [2]:

```
while True:
    try:
        x = int(input('Please enter an interger: '))
        break
    except ValueError:
        print('This is not an interger, please try again!')
    except KeyboardInterrupt:
        print('This is not an interger, please try again!')
```

```
Please enter an interger: 1.1
This is not an interger, please try again!
Please enter an interger: a
This is not an interger, please try again!
Please enter an interger: 2
```

This is how **try** work：
-First, run the subsentence under try（which is the code between **try** and **except**）。
-If there is no exception，then it will ignore **except** and complete the subsentence。
-If this is an exception，then ignore the remaining subsentence before **except**. Then, if the exception is the same type that **except** wanted，then run the code under **except**.
-If the exception is not the same type that **except** wanted，then this exception will be throw outside the **try...except** sentence; If no code handling this exception, then it will be an unhandled exception, the whole program will be stoped and throw this exception.

See more details on https://docs.python.org/zh-cn/3/tutorial/errors.html (https://docs.python.org/zh-cn/3/tutorial/errors.html)