Python try...except

Summary: in this tutorial, you'll learn how to use the Python `try...except` statement to handle exceptions gracefully.

In Python, there're two main kinds of errors: syntax errors and exceptions.

Syntax errors

When you write an invalid Python code, you'll get a syntax error. For example:

```
current = 1
if current < 10
current += 1
```

If you attempt to run this code, you'll get the following error:

```
File "d:/python/try-except.py", line 2
    if current < 10
```

```
                    ^
SyntaxError: invalid syntax
```

In this example, the Python interpreter detected the error at the `if` statement (https://www.pythontutorial.net/python-basics/python-if/) since a colon (`:`) is missing after it.

The Python interpreter shows the file name and line number where the error occurred so that you can fix it.

Exceptions

Even though when your code has valid syntax, it may cause an error during execution.

In Python, errors that occur during the execution are called **exceptions**. The causes of exceptions mainly come from the environment where the code executes. For example:

- Reading a file (https://www.pythontutorial.net/python-basics/python-read-text-file/) that doesn't exist (https://www.pythontutorial.net/python-basics/python-check-if-file-exists/) .
- Connecting to a remote server that is offline.
- Bad user inputs.

When an exception occurs, the program doesn't handle it automatically. This results in an error message.

For example, the following program calculates the sales growth:

```
# get input net sales
print('Enter the net sales for')

previous = float(input('- Prior period:'))
current = float(input('- Current period:'))

# calculate the change in percentage
change = (current - previous) * 100 / previous
```

```
# show the result
if change > 0:
    result = f'Sales increase {abs(change)}%'
else:
    result = f'Sales decrease {abs(change)}%'

print(result)
```

How it works.

- First, prompt users for entering two numbers: the net sales of the prior and current periods.
- Then, calculate the sales growth in percentage and show the result.

When you run the program and enter `120'` as the net sales of the current period, the Python interpreter will issue the following output:

```
Enter the net sales for
- Prior period:100
- Current period:120'
Traceback (most recent call last):
  File "d:/python/try-except.py", line 5, in <module>
    current = float(input('- Current period:'))
ValueError: could not convert string to float: "120'"
```

The Python interpreter showed a traceback that includes detailed information of the exception:

- The path to the source code file (`d:/python/try-except.py`) that caused the exception.
- The exact line of code that caused the exception (`line 5`)
- The statement that caused the exception `current = float(input('- Current period:'))`
- The type of exception `ValueError`
- The error message: `ValueError: could not convert string to float: "120'"`

Because `float()` couldn't convert the string `120'` to a number, the Python interpreter issued a `ValueError` exception.

In Python, exceptions have different types such as `TypeError`, `NameError`, etc.

Handling exceptions

To make the program more robust, you need to handle the exception once it occurs. In other words, you need to catch the exception and inform users so that they can fix it.

A good way to handle this is not to show what the Python interpreter returns. Instead, you replace that error message with a more user-friendly one.

To do that, you can use the Python `try...except` statement:

```
try:
    # code that may cause error
except:
    # handle errors
```

The `try...except` statement works as follows:

- The statements in the `try` clause executes first.
- If no exception occurs, the `except` clause is skipped and the execution of the `try` statement is completed.
- If an exception occurs at any statement in the `try` clause, the **rest of the clause is skipped** and the `except` clause is executed.

The following flowchart illustrates the `try...except` statement:

So to handle exceptions using the `try...except` statement, you place the code that may cause an exception in the `try` clause and the code that handles exception in the `except` clause.

Here's how you can rewrite the program and uses the `try...except` statement to handle the exception:

```
try:
    # get input net sales
    print('Enter the net sales for')

    previous = float(input('- Prior period:'))
    current = float(input('- Current period:'))

    # calculate the change in percentage
    change = (current - previous) * 100 / previous

    # show the result
```

```
    if change > 0:
        result = f'Sales increase {abs(change)}%'
    else:
        result = f'Sales decrease {abs(change)}%'

    print(result)
except:
    print('Error! Please enter a number for net sales.')
```

If you run the program again and enter the net sales which is not a number, the program will issue the message that you specified in the `except` block instead:

```
Enter the net sales for
- Prior period:100
- Current period:120'
Error! Please enter a number for net sales.
```

Catching specific exceptions

When you enter the net sales of the prior period as zero, you'll get the following message:

```
Enter the net sales for
- Prior period:0
- Current period:100
Error! Please enter a number for net sales.
```

In this case, both net sales of the prior and current periods are numbers, the program still issues an error message. Another exception must occur.

The `try...except` statement allows you to handle a particular exception. To catch a selected exception, you place the type of exception after the `except` keyword:

```
try:
    # code that may cause an exception
except ValueError as error:
    # code to handle the exception
```

For example:

```
try:
    # get input net sales
    print('Enter the net sales for')

    previous = float(input('- Prior period:'))
    current = float(input('- Current period:'))

    # calculate the change in percentage
    change = (current - previous) * 100 / previous

    # show the result
    if change > 0:
        result = f'Sales increase {abs(change)}%'
    else:
        result = f'Sales decrease {abs(change)}%'

    print(result)
except ValueError:
    print('Error! Please enter a number for net sales.')
```

When you run a program and enter a string for the net sales, you'll get the same error message.

However, if you enter zero for the net sales of the prior period:

```
Enter the net sales for
- Prior period:0
```

```
- Current period:100
```

... you'll get the following error message:

```
Traceback (most recent call last):
  File "d:/python/try-except.py", line 9, in <module>
    change = (current - previous) * 100 / previous
ZeroDivisionError: float division by zero
```

This time you got the `ZeroDivisionError` exception. This division by zero exception is caused by the following statement:

```
change = (current - previous) * 100 / previous
```

And the reason is that the value of the `previous` is zero.

Handling multiple exceptions

The `try...except` allows you to handle multiple exceptions by specifying multiple `except` clauses:

```
try:
    # code that may cause an exception
except Exception1 as e1:
    # handle exception
except Exception2 as e2:
    # handle exception
except Exception3 as e3:
    # handle exception
```

This allows you to respond to each type of exception differently.

If you want to have the same response to some type of exceptions, you can group them in one except clause:

```
try:
    # code that may cause an exception
except (Exception1, Exception2):
    # handle exception
```

The following example shows how to use the `try...except` to handle the `ValueError` and `ZeroDivisionError` exceptions:

```
try:
    # get input net sales
    print('Enter the net sales for')

    previous = float(input('- Prior period:'))
    current = float(input('- Current period:'))

    # calculate the change in percentage
    change = (current - previous) * 100 / previous

    # show the result
    if change > 0:
        result = f'Sales increase {abs(change)}%'
    else:
        result = f'Sales decrease {abs(change)}%'

    print(result)
except ValueError:
    print('Error! Please enter a number for net sales.')
except ZeroDivisionError:
    print('Error! The prior net sales cannot be zero.')
```

When you enter zero for the net sales of the prior period:

```
Enter the net sales for
- Prior period:0
- Current period:120
```

... you'll get the following error:

```
Error! The prior net sales cannot be zero.
```

It's a good practice to catch other general errors by placing the `catch Exception` block at the end of the list:

```
try:
    # get input net sales
    print('Enter the net sales for')

    previous = float(input('- Prior period:'))
    current = float(input('- Current period:'))

    # calculate the change in percentage
    change = (current - previous) * 100 / previous

    # show the result
    if change > 0:
        result = f'Sales increase {abs(change)}%'
    else:
        result = f'Sales decrease {abs(change)}%'

    print(result)
except ValueError:
    print('Error! Please enter a number for net sales.')
except ZeroDivisionError:
    print('Error! The prior net sales cannot be zero.')
```

```
except Exception as error:
    print(error)
```

Summary

- Use Python `try...except` statement to handle exceptions gracefully.
- Use specific exception in the `except` block as much as possible.
- Use the `except Exception` statement to catch other exceptions.