Errors and Exceptions

There may be three types of error in a Python program.

- Syntax errors: A good editor and Python can help you to figure out syntax errors;
- Runtime errors: The Python interpreter can detect runtime errors and show you where the runtime error occurs;
    - Some examples of Python runtime errors:
        - division by zero
        - performing an operation on incompatible types
        - using an identifier which has not been defined
        - accessing a list element, dictionary value or object attribute which doesn’t exist
        - trying to access a file which doesn’t exist
- Logical errors: You must make use of debugging techniques and tools to find logical errors by yourselves.

Handling Exceptions

Advantages of exception handling:

- Normal codes and codes for handling errors are separate.
- Exceptions can be passed along functions in the stack until they reach a function which handles them. The intermediate functions don’t need to have any error-handling code.
- Exceptions come with lots of useful error information: traceback

Exception handling using Python

- The try-except block can be used to handle exceptions. If no except-blocks match the exception, the program will crash.

```
try:
    age = int(input("Please enter your age: "))
    print("I see that you are {} years old.".format(age))
except ValueError:
    print("Hey, that wasn't a number!")
```

    - The try-except block can handle multiple exceptions simultaneously.

```
try:
    dividend = int(input("Please enter the dividend: "))
    divisor = int(input("Please enter the divisor: "))
    print("{} / {} = {}".format(dividend, divisor, dividend/divisor))
except (ValueError, ZeroDivisionError): # a tuple of exceptions
    print("Oops, something went wrong!")
```

- Python tries to process every statement in the try-block and passes the flow of control to the except-block immediately when a ValueError arises.
    - A ValueError can arise when the input string is not an integer number.

```
try:
    dividend = int(input("Please enter the dividend: "))
    divisor = int(input("Please enter the divisor: "))
    print("{} / {} = {}".format(dividend, divisor, dividend/divisor))
except ValueError:
    print("The divisor and dividend have to be numbers!")
except ZeroDivisionError:
    print("The divisor may not be zero!")
except:
    print("default except must be at the last")
```

- Each exception clause is checked one by one to see if the exception type matches. Once an exception type is matched, the remaining exception clause is ignored. Otherwise, this exception is unhandled and this program will crash.

```
try:
    dividend = int(input("Please enter the dividend: "))
    divisor = int(input("Please enter the divisor: "))
    print("{} / {} = {}".format(dividend, divisor, dividend/divisor))
except ValueError:
    print("The divisor and dividend have to be numbers!")
except ArithmeticError:
    print("An arithmetic error occurs!")
except ZeroDivisionError:
    print("The divisor may not be zero!")
except:
    print("default except must be at the last")
```

- For built-in exceptions, refer to https://docs.python.org/3/library/exceptions.html (https://docs.python.org/3/library/exceptions.html)

```
BaseException
    SystemExit
    KeyboardInterrupt
    GeneratorExit
    Exception
        StopIteration
        StopAsyncIteration
        ArithmeticError
            FloatingPointError
            OverflowError
            ZeroDivisionError
```

In [1]:

```
try:
    dividend = int(input("Please enter the dividend: "))
    divisor = int(input("Please enter the divisor: "))
    print("{} / {} = {}".format(dividend, divisor, dividend/divisor))
except ValueError:
    print("The divisor and dividend have to be numbers!")
except ArithmeticError:
    print("An arithmetic error occurs!")
except ZeroDivisionError:
    print("The divisor may not be zero!")
except:
    print("default except must be at the last")
```

```
Please enter the dividend: 4
Please enter the divisor: 0
An arithmetic error occurs!
```

In [2]:

```
# change the order of handling ZeroDivisionError and ArithmeticError
try:
    dividend = int(input("Please enter the dividend: "))
    divisor = int(input("Please enter the divisor: "))
    print("{} / {} = {}".format(dividend, divisor, dividend/divisor))
except ValueError:
    print("The divisor and dividend have to be numbers!")
except ZeroDivisionError:
    print("The divisor may not be zero!")
except ArithmeticError:
    print("An arithmetic error occurs!")
except:
    print("default except must be at the last")
```

```
Please enter the dividend: 4
Please enter the divisor: 0
The divisor may not be zero!
```

Wrap a small block which may raise exceptions by a try-except block to handle specific exceptions.

In [3]:

```
try:
    dividend = int(input("Please enter the dividend: "))
except ValueError:
    print("The dividend has to be a number!")

try:
    divisor = int(input("Please enter the divisor: "))
except ValueError:
    print("The divisor has to be a number!")

try:
    print("{} / {} = {}".format(dividend, divisor, dividend/divisor))
except ZeroDivisionError:
    print("The dividend may not be zero!")
```

```
Please enter the dividend: 4
Please enter the divisor: 0
The dividend may not be zero!
```

The else and finally Statements

The else-block is executed if the try-block does not raise an exception.

```
try:
    age = int(input("Please enter your age: "))
except ValueError:
    print("Hey, that wasn't a number!")
else:
    print("I see that you are {} years old.".format(age))
```

The finally clause always be executed at the end of the try-except block.

```
try:
    age = int(input("Please enter your age: "))
except ValueError:
    print("Hey, that wasn't a number!")
else:
    print("I see that you are {} years old.".format(age))
finally:
    # cleanup code
    print("It was really nice talking to you. Goodbye!")
```

In [4]:

```
for in_str in ['20abc','20']:
    print('-'*10+' the input is {} '.format(in_str)+'-'*10)
    try:
        age = int(in_str)
    except ValueError:
        print("Hey, that wasn't a number!")
    else:
        print("I see that you are {} years old.".format(age))
    finally:
        # cleanup code
        print("It was really nice talking to you. Goodbye!")
```

```
---------- the input is 20abc ----------
Hey, that wasn't a number!
It was really nice talking to you. Goodbye!
---------- the input is 20 ----------
I see that you are 20 years old.
It was really nice talking to you. Goodbye!
```

Using the Exception Object and Raising Exceptions

Python’s exception objects contain the error type and messages.

In [5]:

```
err_input = '20abc'
try:
    age = int(err_input)
except ValueError as err: # err is not a string but it has an instance method for convertin
    print(err)
```

```
invalid literal for int() with base 10: '20abc'
```

The raise statement can be used to raise an exception. For built-in exceptions, refer to https://docs.python.org/3/library/exceptions.html (https://docs.python.org/3/library/exceptions.html)

```
ValueError
TypeError
NotImplementedError
NameError
…
```

In [6]:

```
err_input = -1
try:
    age = int(err_input)
    if age < 0:
        raise ValueError("{} is not a valid age. Age must be positive or zero.".format(age)
except ValueError as err:
    print("You entered incorrect age input: {}".format(err))
    # raise err # we can pass this exception to the caller function
else:
    print("I see that you are {} years old.".format(age))
```

```
You entered incorrect age input: -1 is not a valid age. Age must be positive
or zero.
```

The try-except statement can be # used to import proper packages.

```
try:
    import CPickle
except ImportError:
    import pickle
```

Handling All Exceptions

- To handle and traceback all exceptions, your main program can be inside a try-except block as follows:

```
      try:
          main_prog()
      except:
          import traceback
          traceback.print_exc()

    Traceback (most recent call last):
      File "I:/7.2.1.1.py", line 27, in <module>
        func()
      File "I:/7.2.1.1.py", line 12, in func
        raise(StopIteration("on purpose"))
    StopIteration: on purpose

    Trackback in the reverse direction
```

In [7]:

```python
def func():
    try:
        for _ in range(1):
            dividend = 4 # int(input("Please enter the dividend: "))
            divisor = 0 # int(input("Please enter the divisor: "))
            raise(StopIteration("on purpose"))
            print("{} / {} = {}".format(dividend, divisor, dividend/divisor))
    except ZeroDivisionError:
        print("The dividend may not be zero!")
    finally:
        print("ByeBye")

    try:
        print("Hi")
    finally:
        print("ByeBye again")

def my_except(t,v,tr):
    print('*************')
    print(t)
    print(v)
    with tr:
        print('---')
        c = tr.tb_name.f_code
        print(c.co_filename)
        print(c.co_name)
        tr = tr.tb_next


import sys

sys.excepthook = my_except

try:
    func()
except :
    import traceback
    traceback.print_exc()
```

```
ByeBye

Traceback (most recent call last):
  File "<ipython-input-7-7c12453b395b>", line 35, in <module>
    func()
  File "<ipython-input-7-7c12453b395b>", line 6, in func
    raise(StopIteration("on purpose"))
StopIteration: on purpose
```

Logging

- The logging module can be used to add logging to a Python. Every message has a level and only the message with a level higher or equal to the level of the program appears in the log.
- The logging level names from the highest level to the lowest level are

- CRITICAL
- ERROR
- WARNING
- INFO
- DEBUG

In [ ]:

```
import logging

# log messages to a file, ignoring anything less severe than ERROR
logging.basicConfig(filename='myprogram.log', level=logging.ERROR)

# these messages should appear in our file
logging.error("The washing machine is leaking!")
logging.critical("The house is on fire!")

# but these ones won't
logging.warning("We're almost out of milk.")
logging.info("It's sunny today.")
logging.debug("I had eggs for breakfast.")

err_input = 'a'
try:
    age = int(err_input)
except ValueError as err:
    # Inside an exception handler, the exception method can be used for logging exceptions.
    logging.exception(err)

logging.shutdown()
```

In [8]:

```
# show the log file
with open('myprogram.log','r') as fp:
    for line in fp:
        print(line,end='')
```

```
ERROR:root:The washing machine is leaking!
CRITICAL:root:The house is on fire!
ERROR:root:invalid literal for int() with base 10: 'a'
Traceback (most recent call last):
  File "<ipython-input-8-60aa0c7736c1>", line 17, in <module>
    age = int(err_input)
ValueError: invalid literal for int() with base 10: 'a'
```