Advance Computer Programming

For RAI

Topic of Error and Exception Handling, Context Management, and Threading

Dr. Sarucha Yanyong, Department of Robotics and AI Engineering

Advance
Computer
Programming
for RAI

Error and Exception Handling

This chapter explores exception and error handling in Python, focusing on its implementation within the language. You will learn about the object model of exception handling, how to throw and catch exceptions, and the process of defining custom exceptions and exception-specific constructs.

What is an error?

```
(catdog) (base) syanyong@syanyong-m1p week05_src % /Users/syanyong/miniconda3/envs/cat
eDrive-sarucha.ya@kmitl.ac.th/My Drive/Teaching/AdvComPro/Week05/week05_src/main.py"
  File "/Users/syanyong/Library/CloudStorage/GoogleDrive-sarucha.ya@kmitl.ac.th/My Dri
    toyota = Car() # create an object / instance an object
    ^
SyntaxError: invalid syntax
(catdog) (base) syanyong@syanyong-m1p week05_src %
```

Error

Handling

This error has propagated beyond the program, resulting in a stack trace of the executed code being displayed. The image shows a Python **SyntaxError: invalid syntax message**, which is likely due to a mistake in the code where the object Car() is being instantiated.

Reference: wiki

Most of the error type in python (1/4)

Error

Handling

```
# Example of a SyntaxError
print("Hello world"  # Missing closing parenthesis
```

```
# Example of a TypeError
result = "10" + 5  # Cannot add a string and an integer
```

```
# Example of a ValueError
number = int("abc")  # Cannot convert a non-numeric string to an integer
```

```
# Example of a NameError
print(variable)  # 'variable' is not defined
```

```
# Example of an IndexError
my_list = [1, 2, 3]
print(my_list[5])  # Index out of range
```


Reference: wiki

Most of the error type in python (2/4)

Error

Handling

```
# Example of a KeyError
my_dict = {"a": 1, "b": 2}
print(my_dict["c"])  # 'c' is not a valid key
```

```
# Example of an AttributeError
my_string = "Hello"
my_string.append("!")  # Strings do not have an append() method
```

```
# Example of an ImportError
import nonexistent_module  # Trying to import a module that doesn't exist
```


Reference: wiki

Most of the error type in python (3/4)

Error

Handling

```
# Example of an IOError (OSError in Python 3)
with open("nonexistent_file.txt") as file:
    content = file.read()  # Trying to open a file that doesn't exist
```

```
# Example of a FileNotFoundError
with open("nonexistent_file.txt") as file:
    content = file.read()  # Raised when a file is not found
```

```
# Example of an UnboundLocalError
def example_function():
    print(variable)  # 'variable' is referenced before assignment
    variable = 10

example_function()
```


Reference: wiki

Most of the error type in python (4/4)

Error

Handling

```
# Example of an IOError (OSError in Python 3)
with open("nonexistent_file.txt") as file:
    content = file.read()  # Trying to open a file that doesn't exist
```

```
# Example of a PermissionError
# Attempting to open a file with insufficient permissions
# Writing to a system file without proper permissions
with open("/etc/hosts", "w") as file:
    file.write("This will fail")
```

```
# Example of a ZeroDivisionError
result = 10 / 0  # Division by zero
```


Reference: wiki

Advance
Computer
Programming
for RAI

Exceptions

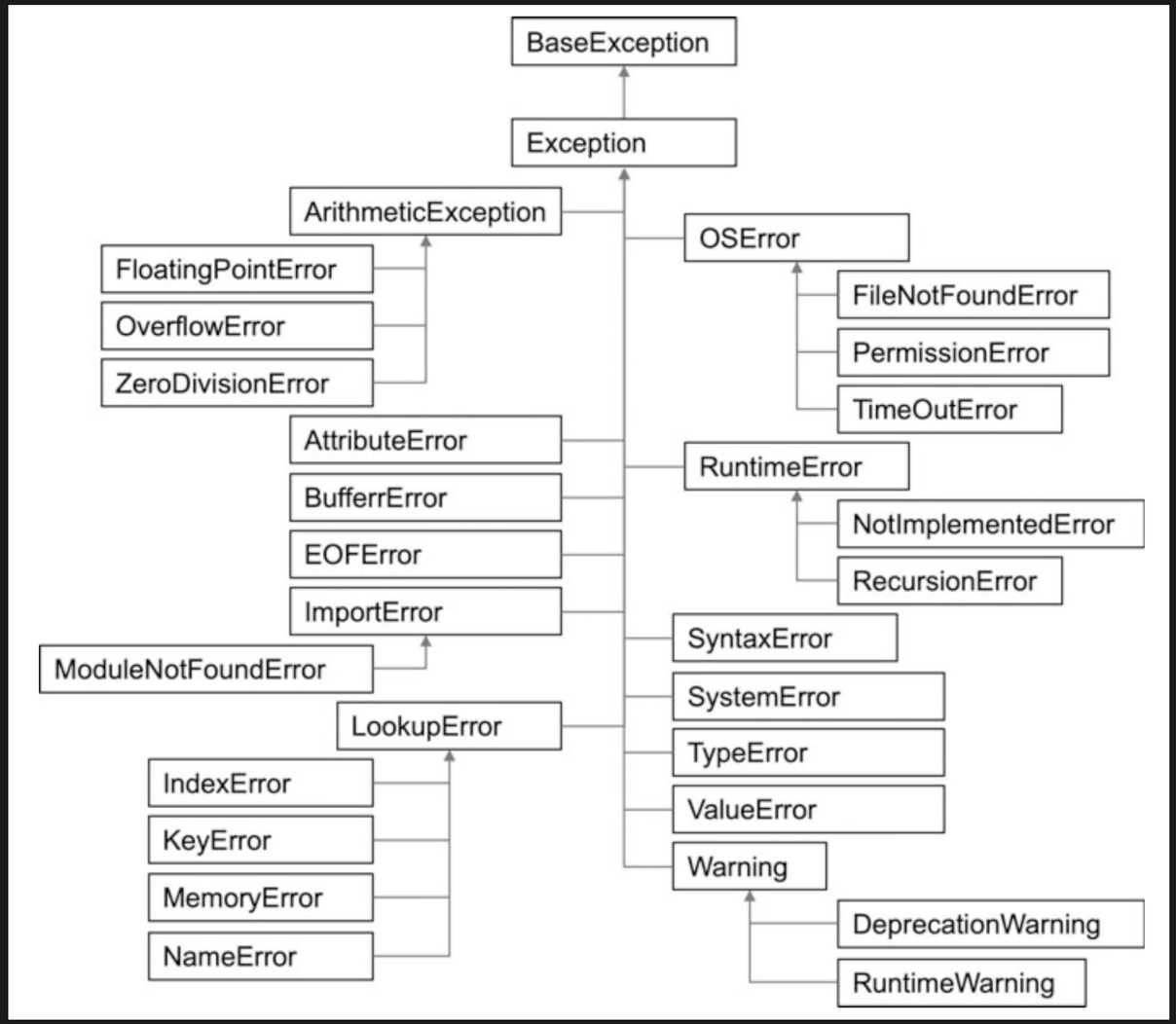
What is an exceptions?

Advance Computer Programming for RAI

What is an Exception?

In Python, an exception is an object that represents an error.

Exceptions are part of a class hierarchy with BaseException at the top, allowing errors to be handled systematically.



Error Handling

Reference: wiki

What is Exception Handling?

Error
Handling

The primary purpose of an exception is to handle such errors at runtime, preventing the program from crashing and allowing for corrective action. For instance, rather than allowing an error like division by zero to disrupt the system, exceptions enable you to prompt the user to correct the input and try again.

Reference: wiki

Terminology

Error Handling

Exception	An error which is generated at runtime
Raising an exception	Generating a new exception
Throwing an exception	Triggering a generated exception
Handling an exception	Processing code that deals with the error
Handler	The code that deals with the error (referred to as the catch block)
Signal	A particular type of exception (such as *out of bounds* or *divide by zero*)

Reference: wiki

Advance
Computer
Programming
for RAI

Handling Exceptions

Using try, except, else, and finally blocks

Error

Exceptions

Basic Exception Handling

```
try:
    result = 10 / 0
except ZeroDivisionError:
    print("You cannot divide by zero!")
else:
    print("The division was successful.")
finally:
    print("This block is executed no matter what.")
```

try

The code inside the try block is executed first. In this case, it attempts to divide 10 by 0, which is an illegal operation in mathematics and raises a ZeroDivisionError.

Reference: wiki

Basic Exception Handling

```
try:
    result = 10 / 0
except ZeroDivisionError:
    print("You cannot divide by zero!")
else:
    print("The division was successful.")
finally:
    print("This block is executed no matter what.")
```

Error

Exceptions

except ZeroDivisionError:
If the code in the try block raises a ZeroDivisionError, this block is executed. It catches the error and prints the message "You cannot divide by zero!". This prevents the program from crashing and allows it to handle the error gracefully.

Reference: wiki

Error

Exceptions

Basic Exception Handling

```
try:
    result = 10 / 0
except ZeroDivisionError:
    print("You cannot divide by zero!")
else:
    print("The division was successful.")
finally:
    print("This block is executed no matter what.")
```

else

The else block is executed only if no exceptions are raised in the try block. Since a ZeroDivisionError is raised in this case, the else block is skipped.

Reference: wiki

Error

Exceptions

Basic Exception Handling

```
try:
    result = 10 / 0
except ZeroDivisionError:
    print("You cannot divide by zero!")
else:
    print("The division was successful.")
finally:
    print("This block is executed no matter what.")
```

finally

The finally block is executed no matter what, whether an exception is raised or not. It is useful for cleanup actions that need to be performed under all circumstances, such as closing files or releasing resources. Here, it simply prints "This block is executed no matter what.".

Reference: wiki

Advance
Computer
Programming
for RAI

Raising Exceptions

Using raise to trigger exceptions

Raising Exceptions

```
def check_age(age):
    if age < 18:
        raise ValueError("Age must be 18 or older.")
    return "Access granted."

try:
    check_age(15)
except ValueError as e:
    print(e)
```

Error

Exceptions

except ValueError as e

- This block is executed if a ValueError is raised within the try block.
- The exception is caught and assigned to the variable e.
- The error message associated with the exception ("Age must be 18 or older.") is then printed to the console using print(e).

Reference: wiki

Advance
Computer
Programming
for RAI

Custom Exceptions

allow you to define your own error types

Why Use Custom Exceptions?

Custom

Exceptions

A custom exception must be used with the raise statement because it is not typically designed to automatically detect errors in the code.

However, the advantage of using a custom exception is that it allows you to handle situations that require detection, even when the code does not generate a standard error.

For example, if a grade is less than 50, an error can be raised.

But why? This approach is one way to collect the error message in the log file.

Reference: wiki

Custom Exceptions

Custom

Exceptions

```python
class NegativeValueError(Exception):
    pass

def calculate_square_root(value):
    if value < 0:
        raise NegativeValueError("Cannot calculate square root of a negative number.")
    return value ** 0.5

try:
    calculate_square_root(-10)
except NegativeValueError as e:
    print(e)
```

explanation

- The error exception name should be defined as class NegativeValueError
- Using raise to assume the error is happen in the code.

Reference: wiki

Advance
Computer
Programming
for RAI

Context Managers

Understand the **with** syntax

Context

management

A context manager in Python is a construct that allows you to allocate and release **resources** precisely when you want to.

When **interfacing** with another system, such as a file, it's essential to not only open the file but also close it properly to prevent errors that could hinder other applications from accessing it.

Reference: wiki

with

Advantage

Traditionally, file handling might look like this:

```
file = open('example.txt', 'r')
try:
    content = file.read()
finally:
    file.close()  # Ensures the file is closed even if an error occurs
```

Shorter using `with`

```
with open('example.txt', 'r') as file:
    content = file.read()
```


Reference: wiki

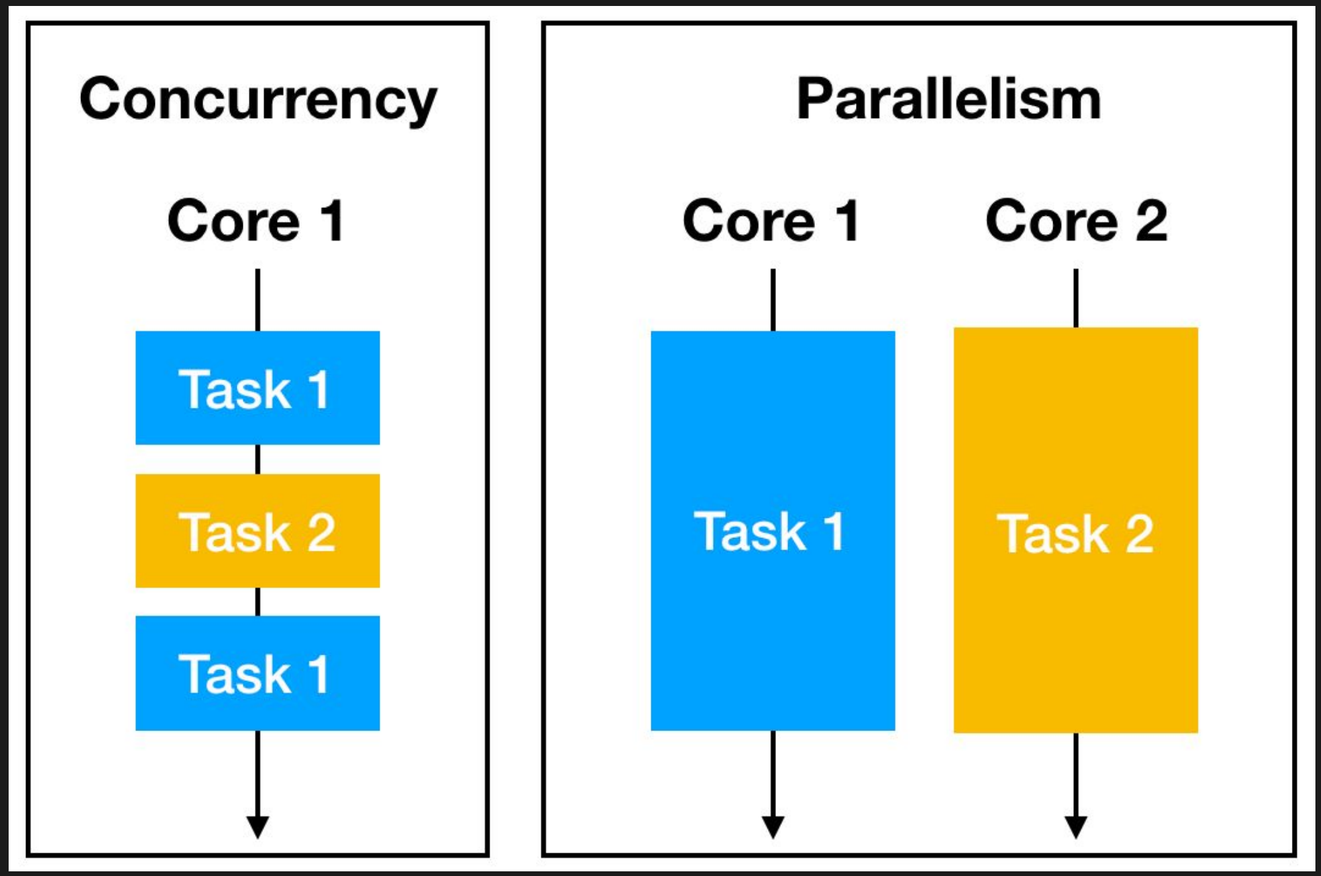
Advance
Computer
Programming
for RAI

Threading

allows a program to run **multiple** operations concurrently in the same process

Threading

What?



concurrency and parallelism or single thread vs multiple threads

Reference: wiki

Threading

What?

Overview of Threading

Threading is a concept that allows a program to run **multiple operations concurrently in the same process**.

It is particularly useful when you want to perform tasks that can run independently of each other, such as handling multiple client connections in a server or performing background tasks without blocking the main application.

Reference: wiki

Threading

Example

```python
import threading
import time

def print_from_thread():
    while True:
        print("This is from the thread")
        time.sleep(1)

# Create and start the thread
thread = threading.Thread(target=print_from_thread)
thread.daemon = True  # This allows the thread to exit when the main program exits
thread.start()

# Main thread while loop
while True:
    print("This is from the main program")
    time.sleep(2)
```


Reference: wiki

Advance Computer Programming for RAI

Done!

Dr. Sarucha Yanyong

Department of Robotics and AI Engineering

School of Engineering, KMITL

EMAIL ADDRESS sarucha.ya@kmitl.ac.th

WEBSITE https://syanyong.github.io

Q/A