

# Errors and exceptions

Inteligencia Artificial en los Sistemas de Control Autónomo  
Máster Universitario en Ingeniería Industrial

Departamento de Automática

## Objectives

1. To be aware of the error handling problem
2. Understand exceptions
3. Handle, create and raise exceptions in Python

## References

Guido van Rossum, “Python Tutorial. Release 3.2.3”, chapter 8

# Table of Contents

1. Definition
2. Handling exceptions
3. Exceptions with arguments
4. Clean-up actions

# Exceptions

## Exception definition (I)

- Errors happen
  - We need a mechanism to handle errors
- Some errors happen before execution (*syntax errors*)
- Others are only detected in execution (*runtime errors*)

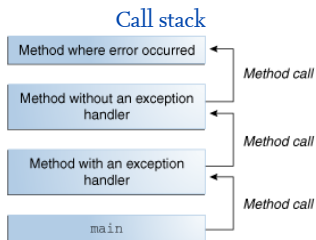
```
>>> while True print('Hello world')
      File "<stdin>", line 1
        while True print('Hello world')
                        ^
```

SyntaxError: invalid syntax

- **Exception:** An error that disrupts the normal execution flow
  - File not found, division by zero, invalid argument, etc
  - Code cannot be executed
  - Elegant solution to handle errors

# Exceptions

## Exception definition (II)

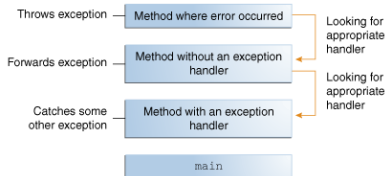


Call stack: Sequence of invoked methods

# Exceptions

## Exception definition (III)

### Exception handling



When an error happens ...

1. Code execution is stopped
2. An exception is thrown
3. The interpreter goes back in the call stack
4. When the interpreter finds an exception handler, it is executed

The exception handler catches the exception, the program finishes otherwise

# Exceptions

## Exception definition (IV)

```
>>> 10 * (1/0)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ZeroDivisionError: division by zero
>>> 4 + spam*3
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'spam' is not defined
>>> '2' + 2
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: Can't convert 'int' object to str implicitly
```

# Exceptions

## Handling exceptions (I)

Handling an exception requires a try-except statement

- **try:** Encloses the vulnerable code
- **catch:** Code that handles the exception

### try-catch statement

```
try :  
    # Risky code  
except ExceptionType1 :  
    # Handle error  
except ExceptionType2 :  
    # Handle error  
except :  
    # Handle errors
```



# Exceptions

## Handling exceptions (II)

### try-catch example

```
1 try:
2     x = int(input("Please enter a number: "))
3 except ValueError:
4     print("Oop!, that was not a number!")
5 except KeyboardInterrupt:
6     print("Got Ctrl-C, good bye!")
```

The exception type contains the error

# Exceptions

## Handling exceptions (III)

### try-catch example

```

1  try:
2      f = open('file.txt')
3      s = f.readline()
4      i = int(s.strip())
5  except IOError as err:
6      print("I/O error: {}".format(err))
7  except ValueError:
8      print("Could not convert data to integer")
9  except:
10     print("Unexpected exception")
11     raise

```

### New Python element

- Raise

# Exceptions

## Exceptions with arguments

Exception arguments: When we need more info

```
1 try:
2     raise Exception("spam", "eggs")
3 except Exception as inst:
4     print(type(inst))
5     print(inst.args)
6     print(inst)
7
8     x, y = inst.args
9     print('x =', x)
10    print('y =', y)
```

```
1 <class 'Exception'>
2 ('spam', 'eggs')
3 ('spam', 'eggs')
4 x = spam
5 y = eggs
```

# Exceptions

## Clean-up actions

Sometimes we need to execute code under all circumstances

- Typically clean-up actions: Close files, database connections, sockets, etc
- The **finally** clause solves this problem

### Example

```
1 try :  
2     raise KeyboardInterrupt  
3 finally :  
4     print("Goodbye , world!")
```