Outline

1. Writing Robust Programs
   - exceptions are unexpected events
   - examples and applications

2. Exceptions in Python
   - raising, handling, defining exceptions
   - exceptions are classes

MCS 260 Lecture 30
Introduction to Computer Science
Jan Verschelde, 17 July 2023
1 Writing Robust Programs
   - exceptions are unexpected events
   - examples and applications

2 Exceptions in Python
   - raising, handling, defining exceptions
   - exceptions are classes
what is an exception?

Goal: make code robust, capable to handle errors.

**Definition (exception)**

An *exception* is an unexpected event that happens during the execution of a program, interrupting the normal flow.

Examples: division by zero in a calculation, wrong user input, or raised by assert statement.

Two phases:

1. the exception is *triggered* (or thrown) by an error, or *raised* explicitly by code in the program;
2. code is executed to *handle* the exception.

Python’s exception mechanism allows programs to handle abnormal situations in a structured way.
Closing Elevator Doors
a real world example

We all use elevators:

- **exception**: many people are not afraid to risk body parts impeding closing doors in order to catch a ride
- **if-else**: attendant with manual closing of doors, doubled up: exterior and interior door

For safety, an elevator without monitoring obstacles between closing doors would be unacceptable.

However, exceptions are for exceptional situations.

Frequent use of exceptions is more expensive than the usual if-else construction.
Writing Robust Programs
- exceptions are unexpected events
- examples and applications

Exceptions in Python
- raising, handling, defining exceptions
- exceptions are classes
some examples of exceptions

- A division by zero occurs.

- When connecting a name to a file:
  - a file opened for reading does not exist, or
  - a file opened for writing does exist.

- Handling the keyboard interrupt.
  
  Do you really want to stop now?
Anytime Algorithms

An anytime algorithm is an algorithm that given some more resources will improve the accuracy of the estimate.

Recall the Monte Carlo method to estimate $\pi$...

Instead of fixing #samples in advance:

1. use of a `while True` loop
2. monitor the progress of the computations via handling of `ctrl+c` interrupt
3. the handler of `KeyboardInterrupt` shows current approximation and asks user to continue or not
raising, handling, and defining exceptions

1. Writing Robust Programs
   - exceptions are unexpected events
   - examples and applications

2. Exceptions in Python
   - raising, handling, defining exceptions
   - exceptions are classes
raising, handling, defining exceptions

1. We can *raise* an exception with `raise`, for example: `raise ZeroDivisionError`.

2. An example is *handled* with the `try-except` statement.

   ```python
   try:
       # code where errors may happen
   except < sequence of exceptions >:
       # code to handle the exception
   ```

3. Exceptions are classes and are defined by inheritance.
the **try**-**except** statement

The syntax of the **try**-**except** statement is

```python
try:
    < code where errors may happen >
except < sequence of exceptions > :
    < code to handle the exception >
```

Three parts in a **try**-**except** statement:

1. **The code following the** `try:` **defines the scope of the associated exception handlers.**

2. **After** `except` **we specify which exceptions to catch.**
   *If the sequence is empty, all exceptions are caught.*

3. **The code to be executed when the exception occurs, follows the** `:` **of the** `except` **.**

Multiple `except` statements may follow one `try` to handle different exceptions differently.
the **try-finally** statement

The syntax of the **try-finally** statement is

```python
try:
    < code where errors may happen >
finally:
    < statements before raising exception >
```

Three parts in a **try-finally** statement:

1. The code following the **try**: defines the scope of the associated exception handlers.
2. Statements after **finally**: will be executed even if an exception happens.
3. After executing the statements after **finally**: the exception will be raised.
Pseudo Code

An anytime algorithm with user controlling the stop criterion:

```
< initialize result >
while True:
    try:
        while True:
            < compute result >
        except KeyboardInterrupt:
            < show result >
        ans = input('continue ? (y/n) ')
        if ans != 'y':
            break
```

The `try-except` stays inside the outer `while True` loop. We leave the outer loop with `break`. 
raising, handling, and defining exceptions

1. Writing Robust Programs
   - exceptions are unexpected events
   - examples and applications

2. Exceptions in Python
   - raising, handling, defining exceptions
   - exceptions are classes
Python’s Exception Hierarchy

exceptions are classes

from the 3.11.4 documentation of Python
Python’s Exception Hierarchy — **OSError**

Observe the **FileExistsError** and **FileNotFoundError** from the 3.11.4 documentation of Python.

```
OSErr0r
 ├── BlockingIOError
 │    ├── BrokenPipeError
 │    │    ├── ConnectionAbortedError
 │    │    │    ├── ConnectionRefusedError
 │    │    │    │    └── ConnectionResetError
 │    ├── FileNotFoundError
 │    ├── InterruptedError
 │    ├── IsADirectoryError
 │    ├── NotADirectoryError
 │    └── PermissionError
 ├── ProcessLookupError
 └── TimeoutError
```
Python’s Exception Hierarchy — Warning

observe the `SyntaxError`, `TypeError`, `ValueError`

from the 3.11.4 documentation of Python
Exercises

1. Write a function that prompts the user for an age. Raise `Exception` when the age is negative.

2. Define an exception `InvalidAge` that has the age as object data attribute.

3. Give an illustration of how to raise the exception defined in previous exercise.